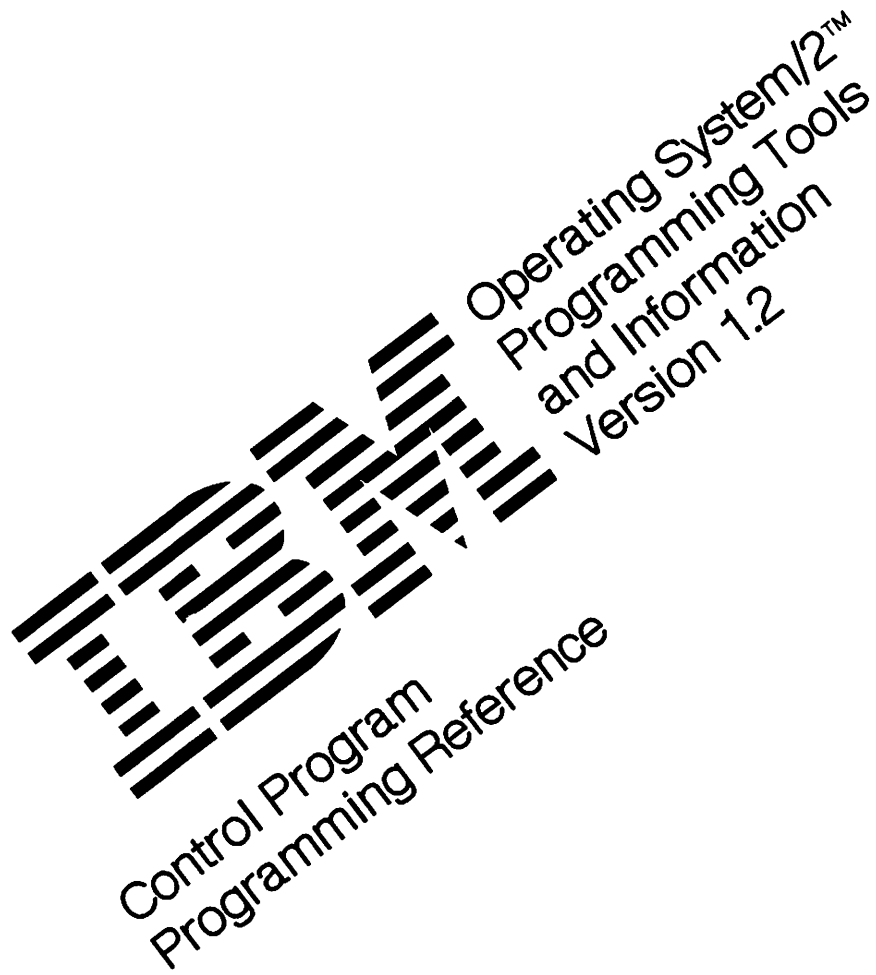


Operating System/2™  
Programming Tools  
and Information  
Version 1.2

Control Program  
Programming Reference

64F0275



**First Edition (September 1989)**

**The following paragraph does not apply to the United Kingdom or any country where such provisions are inconsistent with local law:**

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This publication could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time.

It is possible that this publication may contain reference to, or information about, IBM products (machines and programs), programming, or services that are not announced in your country. Such references or information must not be construed to mean that IBM intends to announce such IBM products, programming, or services in your country.

Any reference to an IBM licensed program or other IBM product in this publication is not intended to state or imply that only IBM's program or other product may be used.

Requests for technical information about IBM products should be made to your IBM Authorized Dealer or your IBM Marketing Representative.

© Copyright International Business Machines Corporation 1986, 1987, 1989

Note to US Government users — Documentation related to Restricted Rights — Use, duplication, or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.

**COPYRIGHT LICENSE:** This publication contains printed sample application programs in source language, which illustrate OS/2 programming techniques. You may copy and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the OS/2 application programming interface.

Each copy of any portion of these sample programs or any derivative work, which is distributed to others, must include a copyright notice as follows: "© (your company name) (year) All Rights Reserved."

---

## Special Notices

The following names, used in this publication, are trademarks or registered trademarks of International Business Machines Corporation in the United States and/or other countries:

IBM  
C/2  
COBOL/2  
FORTRAN/2  
Macro Assembler/2  
Operating System/2  
OS/2  
Presentation Manager  
Systems Application Architecture.

The following names, used in this publication, are trademarks or registered trademarks of other companies:

CodeView                      Intel  
Microsoft Corporation

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license enquiries, in writing, to the IBM Director of Commercial Relations, IBM Corporation, Armonk, NY 10504.



Following is a description of the new function added to the OS/2 Control Program Application Programming Interface (API) by OS/2 Version 1.2.

---

## Support Provided by OS/2 Version 1.2 File Systems

OS/2 Version 1.2 supports the OS/2 File Allocation Table (FAT) and the installable High Performance File System (HPFS), as well as any File System Drivers (FSDs) written for OS/2 Version 1.2. Both OS/2 Version 1.2 file systems provide the following support for the API:

- DOS logical file and directory structure
- DOS format (8.3 filename format) naming conventions
- Universal naming conventions (UNC)
- Partitioning of logical volumes
- Redirection or connection to remote file systems
- Extended Attributes (EAs)
- Global file name character processing.

In addition, the High Performance File System provides support for applications that recognize long file names. (See "Support Specific to FSDs.")

The following API function allows an application to determine the maximum path length allowed by the file system currently in use:

**DosQSysInfo** Returns the maximum path length allowed by the file system. An application calls this function during initialization.

## Creating File Objects with Extended Attributes

Support for applications that are aware of extended attributes and use them to manipulate file objects (files and subdirectories) is provided by the following API functions:

**DosOpen2** Creates a file, opens an existing file, or replaces an existing file. These files may have extended attributes associated with them. In addition to standard attributes, which are set as part of the directory entry, extended attributes (EAs) can be defined and associated with a newly created file.

**DosMkdir2** Creates a subdirectory. EAs can be defined and associated with the subdirectory.

**DosFindFirst2** Finds names of file objects that have particular EAs associated with them. This call is used in conjunction with **DosFindNext** and **DosFindClose**.

**DosEnumAttribute** Identifies EAs associated with an existing file. This call returns the lengths of EA names and values, as well as EA ASCII names. This information is used to calculate the buffer space required to hold EA information returned by a **DosQFileInfo** or **DosQPathInfo** request.

The following API functions allow an application to specify EA information levels for the purpose of querying and setting extended attributes:

**DosQPathInfo** Returns attributes and EA names and values associated with a file object. Query may be made with either the name of a file or subdirectory, or a file handle.

**DosSetPathInfo** Sets attributes and EA names and values for a file or subdirectory.

**DosQFileInfo** Returns attributes and EA names and values associated with an open file. (This function is an enhancement to an existing function.)

**DosSetFileInfo** Sets attributes and EA names and values for an open file. (This function is an enhancement to an existing function.)

Comprehensive guidelines for defining standard EAs as well as application-defined EAs may be found in the "File and Disk Management" chapter of the *Programming Guide*. EA data types and data structures are also discussed.

## Copying File Objects

The following API function supports copying of file objects that may have extended attributes:

**DosCopy** Copies file objects and their associated attributes and EAs from the source directory to the target directory.

## Repetitive Operations on File Objects

The following API function provides global file-name character processing support:

**DosEditName** Searches for and edits the names of file objects. This function is used in conjunction with DosCopy and DosMove to perform repetitive operations on file objects.

## Network I/O Operations

The following API function provides an efficient means of performing I/O to a local or remote FSD or FAT, whose access is being controlled by a network application.

**DosFileIO** Performs combined lock and unlock, seek, and read or write operations on multiple regions within a file.

---

## Support Specific to FSDs

The High Performance File System and any other FSDs written for OS/2 Version 1.2 use an extended standard interface and provide support for applications that recognize long file names. OS/2 Version 1.2 identifies these applications as belonging to the NEWFILES class of application by a bit that is set in the header of the executable file at link time. This bit is not inherited by any child processes started by the application.

The following API functions are provided for network applications that need to control local and remote access to OS/2 FSDs:

**DosFSAttach** Attaches or detaches a drive to or from a local or remote FSD. This function also attaches or detaches a pseudo-character device (such as a named pipe) to the FSD.

**DosQFSAttach** Returns information about a local or remote FSD. This function also returns information about a character or pseudo-character device attached to the FSD.

**DosFSCtl** Provides an extended standard interface between a network application and an FSD.

The High Performance File System also supports caching of directories, data, and file system structures. The following API function allows an application to prevent the corruption of disk files, which can occur if data is still in cache when the system is powered off:

**DosShutdown** Flushes data in cache to disk and prevents further caching of data.

This call is used by the Presentation Manager.

---

## File System Compatibility Considerations

The OS/2 Version 1.2 FAT continues to provide support for OS/2 applications written for Versions 1.0 and 1.1 (as well as any applications written for 1.2) that recognize only DOS format (8.3 filename format) file names. By default, an application of this type is considered a OLDFILES class application. A OLDFILES application exhibits the same behavior as when running in OS/2 Versions 1.0 and 1.1. This behavior is on a per-process basis and extends to any DLL called by the process. Therefore, when a OLDFILES application passes a long file name with a file system request, the request is rejected.

Following is a list of the file system function calls whose acceptance or rejection of long file names is determined by the class of application making the request:

<b>DosCopy</b>	<b>DosMkDir</b>
<b>DosDelete</b>	<b>DosMove</b>
<b>DosExecPgm</b>	<b>DosOpen</b>
<b>DosFindFirst</b>	<b>DosQAppType</b>
<b>DosFindNext</b>	<b>DosQFileMode</b>
<b>DosQFSAttach</b>	<b>DosQPathInfo</b>
<b>DosFSAttach</b>	<b>DosRmDir</b>
<b>DosGetMessage</b>	<b>DosSearchPath</b>
<b>DosGetModHandle</b>	<b>DosSetFileMode</b>
<b>DosGetModName</b>	<b>DosSetPath</b>
<b>DosLoadModule</b>	<b>DosStartSession</b>

The following API functions, which provide EA support and are separate from their counterparts that have similar names and functions for compatibility reasons, also reject a request from a OLDFILES class application that passes a long file name:

**DosOpen2**  
**DosMkDir2**  
**DosFindFirst2**

The only restriction placed on a OLDFILES class of application concerning files with extended attributes is that if critical EAs have been defined and associated with a file, a request to open the file will be rejected. EAs that are defined as critical EAs contain information that is essential to the intended use of the file.

## Family API Considerations

A family application, written to run in the DOS environment as well as the OS/2 environment, also behaves the same as when running in DOS mode for OS/2 Versions 1.0 and 1.1. When a long file name is passed with a file system function request, the name is truncated and is not considered an error. Setting the NEWFILES bit has no effect in DOS mode.

---

## Device Driver Support

The following API functions have been added to support device drivers:

<b>DosDevIOCtl2</b>	Performs control functions on an open device. Supports device drivers that must receive generic IOCTL packets with associated length fields.
<b>DosSMRegisterDD</b>	Registers a device driver with the session manager. The device driver is notified whenever a session switch occurs.

---

## I/O Subsystems Support

The following API function has been added to the OS/2 keyboard subsystem:

<b>KbdGetHWId</b>	Returns the hardware-generated identification value for the attached keyboard.
-------------------	--

The following API function has been added to the OS/2 video subsystem:

<b>VioGlobalReg</b>	Allows global registration of a video subsystem for all full-screen sessions. The video subsystem is notified whenever an application running in a full-screen session completes a video function request.
---------------------	--

---

# Preface

---

## About this Book

The Operating System/2 Version 1.2 (*OS/2 Control Program Programming Reference*) is a detailed technical reference for application programmers creating programs using OS/2 Dos, Kbd, Mou, and Vio system functions. These functions are also called the control program functions. The control program functions carry out such tasks as allocating memory and performing file, keyboard, mouse, and video operations.

Chapter 1 contains important information. You should read it before using this book.

The reference does not give guidance on how to use the calls, nor does it contain information about how the calls are related to each other. It is intended to be used in conjunction with the Operating System/2 Version 1.2 *Programming Guide*.

## Prerequisite Knowledge

The Programming Tools and Information library is intended for professional application developers knowledgeable in at least one programming language in which OS/2 programs can be written. The information in the Programming Tools and Information library assumes you are new to programming with OS/2 and the Presentation Manager. You should understand the OS/2 services available to users. Further information on these can be obtained from the OS/2 Product library.

## Related Publications

The Operating System/2 Version 1.2 *Programming Overview* introduces the programming concepts that you should understand before you begin developing applications to run on OS/2, and describes the set of books, tools, programming aids, and sample programs that make up the OS/2 Programming Tools and Information Library.

Details of publications for OS/2 Version 1.2 and related products are shown in the Library Diagram on the following page.

## OS/2 Product

IBM Operating System/2  
Standard Edition  
Version 1.2

Getting Started  
Using Advanced Features  
Product Information

6024926 3.5" diskette  
6024930 5.25" diskette

## OS/2 Programming Tools and Information

IBM Operating System/2  
Version 1.2

Installation  
Programming Overview  
Programming Guide  
Building Programs  
Dialog Tag Language  
Guide and Reference  
Dialog Manager Guide  
and Reference  
Dialog Manager and  
Dialog Tag Language  
Reference Summary  
Programming Reference  
(3 books)  
Bindings Reference for  
Presentation Manager  
(4 books for COBOL/2,  
FORTRAN/2, C/2, and  
Macro Assembler/2)  
I/O Subsystems and  
Device Support  
(2 books)  
Systems Application  
Architecture  
Common User Access:  
Advanced Interface  
Design Guide

6024929

## Separate Order (no charge)

Keyboards and Code Pages  
6280345

## Available Separately

Command Reference  
6024928

Service Coordinator's  
Guide  
15F2214

## Programming Languages

IBM Basic Compiler/2  
Version 1.0 6280179

IBM Macro Assembler/2  
Version 1.0 6280181

IBM Pascal Compiler/2  
Version 1.0 6280183

IBM FORTRAN/2  
Version 1.02 6280185

IBM COBOL/2  
Version 1.0 6280207

IBM C/2  
Version 1.1 6280284

## Systems Application Architecture

An Overview  
GC26-4341

Writing Applications:  
A Design Guide  
SC26-4362

Common User Access:  
Advanced Interface  
Design Guide  
SC26-4582

Common User Access:  
Basic Interface Design  
Guide  
SC26-4583

## Common Programming Interface

C Reference - Level 2  
SC09-1308

COBOL Reference  
SC26-4354

Dialog Reference  
SC26-4356

FORTRAN Reference  
SC26-4357

Procedures Language  
Reference  
SC26-4358

Presentation Reference  
SC26-4359

### Notes:

1. CodeView™ is available with IBM C/2 Version 1.1 and upon request. An order form is available with the OS/2 Standard Edition Version 1.2 Product. It is also compatible with the Macro Assembler/2, Pascal Compiler/2, and BASIC Compiler/2 listed above.

2. *IBM C/2 Version 1.1 is available on 5.25 in. diskette. An order form is available in the C/2 package.*
3. *See your IBM representative for ordering information.*

---

# Organization of this Book

## **Chapter 1, “Introduction” on page 1-1**

This chapter contains important information about the following:

- Notation conventions
- Conventions used in Function Descriptions
- Format of Call Descriptions

You should read it before using this book.

## **Chapter 2, “Control Program Function Calls” on page 2-1**

## **Chapter 3, “Keyboard Function Calls” on page 3-1**

## **Chapter 4, “Mouse Function Calls” on page 4-1**

## **Chapter 5, “Video Function Calls” on page 5-1**

## **Appendix A, “Errors Returned from Base OS/2 Calls” on page A-1**

## **Index**

# Contents

<b>Chapter 1. Introduction</b>	<b>1-1</b>
Conventions Used in Function Descriptions	1-1
<b>Chapter 2. Control Program Function Calls</b>	<b>2-1</b>
DosAllocHuge — Allocate Huge Memory	2-2
DosAllocSeg — Allocate Segment	2-5
DosAllocShrSeg — Allocate Named Shared Segment	2-8
DosBeep — Generate Sound From Speaker	2-10
DosBufReset — Commit File's Cache Buffers	2-11
DosCallback — Privilege Level 2 Call to Privilege Level 3 Routine	2-13
DosCallNmPipe — Perform Procedure Call Transaction	2-14
DosCaseMap — Perform Case Mapping	2-16
DosChDir — Change Current Directory	2-18
DosChgFilePtr — Move File Read/Write Pointer	2-20
DosCLIAccess — Request CLI/STI Privilege	2-22
DosClose — Close File Handle	2-23
DosCloseQueue — Close Queue	2-25
DosCloseSem — Close System Semaphore	2-26
DosConnectNmPipe — Connect Named Pipe	2-28
DosCopy — Copy File	2-30
DosCreateCSAlias — Create CS Alias	2-32
DosCreateQueue — Create Queue	2-34
DosCreateSem — Create System Semaphore	2-36
DosCreateThread — Create Another Thread of Execution	2-39
DosCwait — Wait for Child Termination	2-42
DosDelete — Delete File	2-45
DosDevConfig — Get Device Configuration	2-47
DosDevIOCtl — Performs I/O Control for Devices	2-49
DosDevIOCtl2 — Performs I/O Control for Devices	2-51
DosDisConnectNmPipe — Disconnect Named Pipe	2-53
DosDupHandle — Duplicate File Handle	2-54
DosEditName — Search for and Edit Names of File Objects	2-56
DosEnterCritSec — Enter Critical Section of Execution	2-58
DosEnumAttribute — Enumerate the extended attributes	2-59
DosErrClass — Classify Error Codes	2-62
DosError — Enable Hard Error Processing	2-65
DosExecPgm — Execute Program	2-67
DosExit — Exit Program	2-73
DosExitCritSec — Exit Critical Section of Execution	2-76
DosExitList — Maintains Routine List for Process Termination	2-77
DosFileIO — Perform File I/O	2-79
DosFileLocks — Manages File Locking	2-82
DosFindClose — Close Find Handle	2-85
DosFindFirst — Find First Matching File Object	2-86
DosFindFirst2 — Find First Matching File Object with Extended Attributes	2-91
DosFindNext — Find Next Matching File	2-99
DosFlagProcess — Set Process External Event Flag	2-103
DosFreeModule — Free Dynamic Link Module	2-106
DosFreeResource — Free Resource	2-108
DosFreeSeg — Free Segment	2-109
DosFSAttach — Creates and Destroys FSD Connections	2-111
DosFSCtl — Provide File System Control	2-113
DosFSRamSemClear — Release a Fast-Safe RAM Semaphore	2-115
DosFSRamSemRequest — Request Safe RAM Semaphore	2-117
DosGetCollate — Get Collate Table	2-120
DosGetCp — Get Process Code Page	2-122
DosGetCtryInfo — Get Country Information	2-124
DosGetDateTime — Get Current Date and Time	2-128
DosGetDBCSEv — Get DBCS Environmental Vector	2-131



DosGetEnv	Get Address of Process Environment String	2-133
DosGetHugeShift	Get Shift Count	2-135
DosGetInfoSeg	Get Address of System Variables Segment	2-136
DosGetMachineMode	Return Current Mode of Processor	2-142
DosGetMessage	Retrieve System Message with Variable Text	2-143
DosGetModHandle	Get Dynamic Link Module Handle	2-146
DosGetModName	Get Dynamic Link Module Name	2-147
DosGetPID	Return Process ID	2-148
DosGetPPID	Get a Process's Parent's PID	2-151
DosGetProcAddr	Get Dynamic Link Procedure Address	2-153
DosGetPrty	Get Process's Priority	2-154
DosGetResource	Get Resource Segment Selector	2-157
DosGetResource2	Get Resource Address	2-159
DosGetSeg	Access Segment	2-161
DosGetShrSeg	Access Shared Segment	2-162
DosGetVersion	Get OS/2 Version Number	2-163
DosGiveSeg	Give Access to Segment	2-165
DosHoldSignal	Disable/Enable Signals	2-166
DosInsMessage	Insert Variable Text Strings In Message	2-169
DosKillProcess	Terminate Process	2-171
DosLoadModule	Load Dynamic Link Module	2-174
DosLockSeg	Lock Segment in Memory	2-176
DosMakeNmPipe	Create Named Pipe	2-177
DosMakePipe	Create Pipe	2-180
DosMemAvail	Get Size of Largest Free Memory Block	2-181
DosMkDir	Make Subdirectory	2-182
DosMkDir2	Make Subdirectory and Define Extended Attributes	2-183
DosMonClose	Close Connection to Device Monitor	2-187
DosMonOpen	Open Connection to Device Monitor	2-188
DosMonRead	Read Input from Monitor Structure	2-189
DosMonReg	Register Set of Buffers as Monitor	2-191
DosMonWrite	Write Output to Monitor Structure	2-193
DosMove	Move a File	2-194
DosMuxSemWait	Wait for One of N Semaphores to Clear	2-196
DosNewSize	Change File Size	2-200
DosOpen	Open File	2-201
DosOpen2	Open File	2-207
DosOpenQueue	Open Queue	2-214
DosOpenSem	Open Existing System Semaphore	2-215
DosPeekNmPipe	Peek Named Pipe	2-218
DosPeekQueue	Peek Queue	2-220
DosPFSActivate	Activate Font	2-222
DosPFSCloseUser	Close Font User Interface	2-224
DosPFSInit	Initialize Code Page and Font	2-225
DosPFSQueryAct	Query Active Font	2-227
DosPFSVerifyFont	Verify Font	2-229
DosPhysicalDisk	Get Information about Partitionable Disk	2-231
DosPortAccess	Request Port Access	2-233
DosPtrace	Provides an Interface for Program Debugging	2-235
DosPurgeQueue	Purge Queue	2-241
DosPutMessage	Output Message Text to Indicated Handle	2-242
DosQAppType	Return the Application Type	2-243
DosQCurDir	Query Current Directory	2-245
DosQCurDisk	Query Current Disk	2-247
DosQFHandleState	Query File Handle State	2-248
DosQFileInfo	Query File Information	2-250
DosQFileMode	Query File Mode	2-255
DosQFSAttach	Query Attached FSD Information	2-257
DosQFSInfo	Query File System Information	2-260
DosQHAndType	Query Handle Type	2-263
DosQNmPHAndState	Query Named Pipe State	2-265
DosQNmPipeInfo	Query Named Pipe Info	2-267
DosQNmPipeSemState	Query Named Pipe Operations	2-269

DosQPathInfo	— Query File or Subdirectory for Information	2-271
DosQSysInfo	— Query System Variable Information	2-277
DosQueryQueue	— Query Size of Queue	2-278
DosQVerify	— Query Verify Setting	2-279
DosR2StackRealloc	— Reallocate Privilege Level 2 Stack	2-280
DosRead	— Read from File	2-281
DosReadAsync	— Asynchronous Read from File	2-283
DosReadQueue	— Read from Queue	2-285
DosReallocHuge	— Change Huge Memory Size	2-287
DosReallocSeg	— Change Segment Size	2-289
DosResumeThread	— Restart Thread	2-291
DosRmdir	— Remove Subdirectory	2-293
DosScanEnv	— Scan an Environment Segment	2-294
DosSearchPath	— Search Path for File Name	2-296
DosSelectDisk	— Select Default Drive	2-299
DosSelectSession	— Select Foreground Session	2-300
DosSemClear	— Clear Semaphore	2-301
DosSemRequest	— Request Semaphore	2-303
DosSemSet	— Set Semaphore Owned	2-306
DosSemSetWait	— Set Semaphore and Wait for Next Clear	2-309
DosSemWait	— Wait for Semaphore To Clear	2-313
DosSendSignal	— Send SIGINTR or SIGBREAK Signal	2-317
DosSetCp	— Set Code Page	2-320
DosSetDateTime	— Set Current Date and Time	2-322
DosSetFHandState	— Set File Handle State	2-324
DosSetFileInfo	— Set File Information	2-326
DosSetFileMode	— Set File Mode	2-331
DosSetFSInfo	— Set File System Information	2-333
DosSetMaxFH	— Set Maximum File Handles	2-335
DosSetNmPHandState	— Set Named Pipe Handle State	2-336
DosSetNmPipeSem	— Set Named Pipe Semaphore	2-338
DosSetPathInfo	— Set Information for a File or Directory	2-340
DosSetProcCp	— Set Process Code Page	2-344
DosSetPrtty	— Set Process Priority	2-345
DosSetSession	— Set Session Status	2-349
DosSetSigHandler	— Set Signal Handler	2-352
DosSetVec	— Establish Handler for Exception Vector	2-356
DosSetVerify	— Set/Reset Verify Switch	2-358
DosSizeSeg	— Get Size of a Segment	2-359
DosShutdown	— Shutdown File Systems for Power Off	2-360
DosSleep	— Delay Process Execution	2-361
DosSMRegisterDD	— Register Session Switch Notification	2-364
DosStartSession	— Start Session	2-366
DosStopSession	— Stop Session	2-373
DosSubAlloc	— Suballocate Memory within Segment	2-375
DosSubFree	— Free Memory Suballocated Within Segment	2-376
DosSubSet	— Initialize or Set Allocated Memory	2-377
DosSuspendThread	— Suspend Thread Execution	2-379
DosTimerAsync	— Start Asynchronous Time Delay	2-381
DosTimerStart	— Start Periodic Interval Timer	2-383
DosTimerStop	— Stop Interval Timer	2-385
DosTransactNmPipe	— Perform Transaction	2-387
DosUnlockSeg	— Unlock Segment	2-389
DosWaitNmPipe	— Wait Named Pipe Instance	2-390
DosWrite	— Synchronous Write to File	2-391
DosWriteAsync	— Asynchronous Write to File	2-394
DosWriteQueue	— Write to Queue	2-396
 <b>Chapter 3. Keyboard Function Calls</b>		<b>3-1</b>
KbdCharIn	— Read Character, Scan Code	3-2
KbdClose	— Close a Logical Keyboard	3-5
KbdDeRegister	— Deregister Keyboard Subsystem	3-6
KbdFlushBuffer	— Flush Keystroke Buffer	3-7

KbdFreeFocus	— Free Keyboard Focus	3-8
KbdGetCp	— Get Loaded Code Page ID	3-9
KbdGetFocus	— Get Keyboard Focus	3-10
KbdGetHWId	— Query Keyboard Hardware ID	3-11
KbdGetStatus	— Get Keyboard Status	3-13
KbdOpen	— Open a Logical Keyboard	3-16
KbdPeek	— Peek at Character, Scan Code	3-17
KbdRegister	— Register Keyboard Subsystem	3-20
KbdSetCp	— Set the Code Page	3-22
KbdSetCustXt	— Set Custom Translate Table	3-24
KbdSetFgnd	— Set Foreground Keyboard Priority	3-25
KbdSetStatus	— Set Keyboard Status	3-26
KbdStringIn	— Read Character String	3-29
KbdSynch	— Synchronize Keyboard Access	3-31
KbdXlate	— Translate Scan Code	3-32
<b>Chapter 4. Mouse Function Calls</b>		<b>4-1</b>
MouClose	— Close Mouse Device	4-2
MouDeRegister	— Deregister a Subsystem	4-3
MouDrawPtr	— Mouse Draw Pointer	4-4
MouFlushQue	— Flush Mouse Queue	4-5
MouGetDevStatus	— Get Mouse Device Status	4-6
MouGetEventMask	— Get Mouse Event Mask	4-7
MouGetNumButtons	— Get Number of Mouse Buttons	4-8
MouGetNumMickey	— Get Number of Mouse Mickeys	4-9
MouGetNumQueEl	— Get Event Queue Status	4-10
MouGetPtrPos	— Query Mouse Pointer Position	4-12
MouGetPtrShape	— Get Pointer Shape	4-14
MouGetScaleFact	— Get Mouse Scaling Factors	4-17
MouInitReal	— Initialize DOS mode	4-19
MouOpen	— Open Mouse Device	4-21
MouReadEventQue	— Read Mouse Event Queue	4-23
MouRegister	— Register a Subsystem	4-26
MouRemovePtr	— Remove Mouse Pointer	4-29
MouSetDevStatus	— Set Mouse Device Status	4-31
MouSetEventMask	— Set Mouse Event Mask	4-33
MouSetPtrPos	— Set Mouse Pointer Position	4-35
MouSetPtrShape	— Set Mouse Pointer Shape	4-37
MouSetScaleFact	— Set Mouse Scaling Factor	4-40
MouSynch	— Get Synchronous Access	4-42
<b>Chapter 5. Video Function Calls</b>		<b>5-1</b>
VioDeRegister	— DeRegister Video Subsystem	5-2
VioEndPopUp	— Deallocate Pop-up Display Screen	5-3
VioGetAnsi	— Get ANSI Status	5-4
VioGetBuf	— Get Logical Video Buffer	5-5
VioGetConfig	— Get Video Configuration	5-7
VioGetCp	— Query Video Code Page	5-11
VioGetCurPos	— Get Cursor Position	5-12
VioGetCurType	— Get Cursor Type	5-13
VioGetFont	— Get Font	5-15
VioGetMode	— Get Display Mode	5-17
VioGetPhysBuf	— Get Physical Display Buffer	5-20
VioGetState	— Get Video State	5-22
VioGlobalReg	— Globally register a video subsystem	5-27
VioModeUndo	— Restore Mode Undo	5-30
VioModeWait	— Restore Mode Wait	5-32
VioPopUp	— Allocate a Pop-up Display Screen	5-34
VioPrtSc	— Print Screen	5-37
VioPrtScToggle	— Toggle Print Screen	5-38
VioReadCellStr	— Read Char/Attr String	5-39
VioReadCharStr	— Read Character String	5-41
VioRegister	— Register Video Subsystem	5-43

VioSavRedrawUndo — Screen Save Redraw Undo	5-46
VioSavRedrawWait — Screen Save Redraw Wait	5-48
VioScrLock — Lock Screen	5-50
VioScrollDn — Scroll Screen Down	5-52
VioScrollLf — Scroll Screen Left	5-54
VioScrollRt — Scroll Screen Right	5-56
VioScrollUp — Scroll Screen Up	5-58
VioScrUnLock — Unlock Screen	5-60
VioSetAnsi — Set ANSI On or Off	5-61
VioSetCp — Set Code Page	5-62
VioSetCurPos — Set Cursor Position	5-64
VioSetCurType — Set Cursor Type	5-65
VioSetFont — Set Font	5-67
VioSetMode — Set Display Mode	5-69
VioSetState — Set Video State	5-74
VioShowBuf — Display Logical Buffer	5-79
VioWrtCellStr — Write Char/Attr String	5-80
VioWrtCharStr — Write Character String	5-82
VioWrtCharStrAtt — Write Char String with Attr	5-84
VioWrtNAttr — Write N Attributes	5-86
VioWrtNCell — Write N Char/Attrs	5-88
VioWrtNChar — Write N Characters	5-90
VioWrtTTY — Write TTY String	5-92
<b>Appendix A. Errors Returned from Base OS/2 Calls</b>	<b>A-1</b>
Numeric Order	A-1
Alphabetic Order	A-7
<b>Index</b>	<b>X-1</b>



---

# Chapter 1. Introduction

This chapter contains important information. You should read it before using this book.

The purpose of this reference is to provide information about control program (base system) function calls, messages, constants, and so on. The control program function calls include four function groups. These function groups provide the basic operating-system features of OS/2. The four function groups are:

<b>Dos</b>	The Dos function calls can be used in full-screen and Presentation Manager sessions to perform basic operating-system operations, such as file input/output, memory allocation, and thread and process creation/control/communication.
<b>Kbd</b>	The Kbd function calls can be used in full-screen sessions to perform keyboard operations.
<b>Mou</b>	The Mou function calls can be used in full-screen sessions to perform mouse operations.
<b>Vio</b>	The Vio function calls can be used in full-screen sessions to perform video operations.

The bindings for both *IBM C/2* and *IBM Macro Assembler/2* languages are given at the end of each function call. An example in C/2 language is also shown at the end of some function calls.

---

## Conventions Used in Function Descriptions

The documentation of each function contains these sections:

<b>Function name</b>	The function name is listed in alphabetic order at the top of each page followed by a brief description of the function.						
<b>Icon</b>	<p>An icon is at the top of each page opposite the function name, where applicable. The icon is used to identify certain characteristics or exceptions. The absence of an icon indicates there are no restrictions.</p> <table><tr><td><b>FAPI</b></td><td>This indicates that the function is a family API function. A family API function can be used in programs for either OS/2 or DOS.</td></tr><tr><td><b>xPM</b></td><td>This indicates that the function is not applicable to Presentation Manager (some Dos, all Kbd, and all Mou calls).</td></tr><tr><td><b>xWPM</b></td><td>This indicates that the function is not applicable to windowable or Presentation Manager calls (some VIO calls).</td></tr></table>	<b>FAPI</b>	This indicates that the function is a family API function. A family API function can be used in programs for either OS/2 or DOS.	<b>xPM</b>	This indicates that the function is not applicable to Presentation Manager (some Dos, all Kbd, and all Mou calls).	<b>xWPM</b>	This indicates that the function is not applicable to windowable or Presentation Manager calls (some VIO calls).
<b>FAPI</b>	This indicates that the function is a family API function. A family API function can be used in programs for either OS/2 or DOS.						
<b>xPM</b>	This indicates that the function is not applicable to Presentation Manager (some Dos, all Kbd, and all Mou calls).						
<b>xWPM</b>	This indicates that the function is not applicable to windowable or Presentation Manager calls (some VIO calls).						
<b>Parameters</b>	<p>Each parameter is listed with its C/2 language data type, parameter type, and a brief description.</p> <ul style="list-style-type: none"><li>All data types are written in uppercase. A data type 'Pxxxxxx' implicitly defines a pointer to the data type 'xxxxxx'. The C/2 language data types, structures, and constants are defined in the OS2DEF.H and BSE*.H include files. The include files are in the TOOLKT12\C\INCLUDE directory.</li><li>The term NULL applied to a parameter indicates the presence of the parameter, but with no value.</li><li>There are three parameter types:<table><tr><td><b>Input</b></td><td>Specified by the programmer.</td></tr><tr><td><b>Output</b></td><td>Returned by OS/2.</td></tr><tr><td><b>Input/Output</b></td><td>Specified by the programmer and modified by OS/2.</td></tr></table></li><li>A brief description is provided with each parameter. Where appropriate, restrictions are also included. In some cases, the parameter points to a structure that is also defined in the parameter description.</li></ul>	<b>Input</b>	Specified by the programmer.	<b>Output</b>	Returned by OS/2.	<b>Input/Output</b>	Specified by the programmer and modified by OS/2.
<b>Input</b>	Specified by the programmer.						
<b>Output</b>	Returned by OS/2.						
<b>Input/Output</b>	Specified by the programmer and modified by OS/2.						
<b>rc</b>	A list of possible return codes or errors (where appropriate) is included in this section. Some functions do not have return codes or error messages. Refer to Appendix A for a complete list of all return codes and their descriptions. The return codes are listed in Appendix A in alphabetic order and in numeric order.						

<b>Remarks</b>	The remarks section contains additional information about the function, where required.
<b>Named Pipe Considerations</b>	This section contains additional information about the function when using Named Pipes, where required.
<b>Family API Considerations</b>	This section contains additional information about the function when the application is intended for DOS, where required.
<b>PM Considerations</b>	This section contains additional information about the function when using Presentation Manager, where required.
<b>Bindings</b>	C/2 and Macro Assembler/2 language bindings are provided with each function. The bindings are derived from the include files. The minimum include statement ( <code>#define INCL_xxxx</code> for C/2 and <code>INCL_xxxx EQU ?</code> for Macro Assembler/2) is provided in each binding.
<b>Example</b>	An example is shown in C/2 language for some functions.
<b>Programming Note:</b> The functions in this book are named in mixed-case for readability, but are known to the system as uppercase character strings. If you are using a compiler that generates a mixed-case external name, you should code the OS/2 function calls in uppercase.	

---

## Chapter 2. Control Program Function Calls

This chapter reflects the Dos API interface of OS/2 only. The Dos function calls can be used in full-screen and Presentation Manager sessions to perform basic operating-system operations, such as file input/output, memory allocation, and thread and process creation/control/communication.

### Notes:

1. Calls marked xPM are not supported by Presentation Manager, and must not be used by Presentation Manager applications. An error code is returned if any of these calls are issued.
2. Calls marked xWPM are not windowable and are not supported by Presentation Manager. They can be used in OS/2 mode.
3. Calls marked FAPI are present in the Family API.
4. Those calls without an icon are:
  - Windowable
  - Presentation Manager
  - Full-screen
  - Not Family API.



# DosAllocHuge — Allocate Huge Memory

FAPI

This call allocates multiple segments as a huge block of memory.

<b>DosAllocHuge (NumSeg, Size, Selector, MaxNumSeg, AllocFlags)</b>
---

## Parameters

**NumSeg** (*USHORT*) — input

Number of 65536-byte segments to be allocated.

**Size** (*USHORT*) — input

Number of bytes to be allocated in the last (non-65536-byte) segment. A value of zero indicates none.

**Selector** (*PSEL*) — output

Address where the selector of the first segment allocated is returned.

**MaxNumSeg** (*USHORT*) — input

Maximum number of 65536-byte segments this object occupies as a result of any subsequent `DosReallocHuge` (see “`DosReallocHuge` — Change Huge Memory Size” on page 2-287). If `MaxNumSeg` is 0, OS/2 assumes this segment will never be increased by `DosReallocHuge` beyond its original size, though it may be decreased. This value is ignored in the DOS mode.

**AllocFlags** (*USHORT*) — input

Bit indicators describing the characteristics of the segment allocated. The bits that can be set and their meanings are:

Bit	Description
15–4	Reserved and must be set to zero.
3	If segment is shared, it can be decreased in size by <code>DosReallocHuge</code> .
2	Segment may be discarded by the system in low memory situations.
1	Segment is shareable through <code>DosGetSeg</code> .
0	Segment is shareable through <code>DosGiveSeg</code> .

**rc** (*USHORT*) — return

Return code descriptions are:

0	NO_ERROR
8	ERROR_NOT_ENOUGH_MEMORY
87	ERROR_INVALID_PARAMETER
212	ERROR_LOCKED

## Remarks

`DosAllocHuge` allows a process to allocate a large amount of memory by telling the system how many 64KB segments it needs and whether it requires an additional partial segment. The system allocates the memory, which is movable and swappable, and returns a selector to the first segment. When this selector is used with a call, the requested function is performed for the entire block of memory.

Each segment of a huge memory allocation has a unique selector. To determine the remaining selectors of a huge memory allocation, issue `DosGetHugeShift`, which returns a shift count. To compute the next sequential selector, take the value 1 and shift it left by the number of bits specified in shift count. Use the resulting value as an increment to add to the previous selector, using the selector returned by `DosAllocHuge` as the first selector. For example:

- Assume `DosAllocHuge` is issued with `NumSeg` equal to 3, and that the number 63 is returned for the selector of the first segment.
- If `DosGetHugeShift` returns a shift count of 4, shifting the value “1” by this amount results in an increment of 16.
- Adding this increment to selector number 63 produces 79 for the second selector. Adding the same increment to selector number 79 yields 95 for the third selector.

# DosAllocHuge — Allocate Huge Memory

Like single segment memory allocated with `DosAllocSeg`, huge memory can be designated as shareable by other processes and discardable by the system when no longer needed. Allocating a huge block of memory as discardable automatically locks the memory for use by the caller. When one segment of a huge allocation is discarded by the system, this forces the discard of all the other segments. See `DosAllocSeg` for more information relating to discardable and shared segments.

Applications should be discretionary in claiming large memory when doing so can impair system performance. To test system memory availability, issue `DosMemAvail`. This call returns the size of the largest block of unallocated memory. Although this value can change at any time because of system activity, it can provide a good indication of the system memory state.

Memory allocated by `DosAllocHuge` is freed by `DosFreeSeg`. One call to `DosFreeSeg`, passing the selector returned from a `DosAllocHuge`, frees all of the memory allocated.

**Note:** This request may be issued from privilege level 2. However, the segment is allocated as a privilege level 3 segment.

## Family API Considerations

Some options operate differently in the DOS mode than in the OS/2 mode. Therefore, the following considerations apply to `DosAllocHuge` when coding for the DOS mode:

- Requested size value is rounded up to the next paragraph (16-byte)
- Selector is the actual segment address allocated.

## C Language

```
#define INCL_DOSMEMMGR

USHORT rc = DosAllocHuge(NumSeg, Size, Selector, MaxNumSeg, AllocFlags);

USHORT      NumSeg;      /* Number of 65536-byte segments */
USHORT      Size;        /* Number of bytes in last segment */
PSEL        Selector;    /* The first Selector allocated (returned) */
USHORT      MaxNumSeg;    /* Max number of 65536-byte segments */
USHORT      AllocFlags;   /* Allocation flags */

USHORT      rc;          /* return code */
```

## Assembler Language

```
EXTRN DosAllocHuge:FAR
INCL_DOSMEMMGR EQU 1

PUSH WORD NumSeg      ;Number of 65536-byte segments
PUSH WORD Size        ;Number of bytes in last segment
PUSH@ WORD Selector   ;The first Selector allocated (returned)
PUSH WORD MaxNumSeg   ;Max number of 65536-byte segments
PUSH WORD AllocFlags  ;Allocation flags
CALL DosAllocHuge
```

Returns WORD

# DosAllocHuge — Allocate Huge Memory

FAPI

## Example

This example requests a block of memory with 4 segments, the last segment having 1,040 bytes. The block of memory will never be larger than 8 segments. The memory can be shared with DosGiveSeg API calls. The system can discard the memory if it needs too.

```
#define INCL_DOSMEMMGR

#define NUMBER_OF_SEGMENTS 4
#define BYTES_IN_LAST_SEGMENT 1040
#define MAXIMUM_SEG_SIZE 8
#define ALLOC_FLAG SEG_GIVEABLE | SEG_DISCARDABLE

SEL    Selector;
USHORT rc;

rc = DosAllocHuge(NUMBER_OF_SEGMENTS, /* # of 65536-byte segments */
                  BYTES_IN_LAST_SEGMENT, /* # of bytes in last segment */
                  &Selector, /* The 1st selector allocated */
                  MAXIMUM_SEG_SIZE, /* Max number of segments */
                  ALLOC_FLAG); /* Allocation flags */
```

This call allocates a segment of memory to a requesting process.

**DosAllocSeg (Size, Selector, AllocFlags)**

## Parameters

**Size** (*USHORT*) – input

Number of bytes requested. The value specified must be less than or equal to 65535. A value of zero indicates 65536 bytes.

**Selector** (*PSEL*) – output

Address where the selector of the segment allocated is returned.

**AllocFlags** (*USHORT*) – input

Bit indicators describing the characteristics of the segment being allocated. The bits that can be set and their meanings are:

Bit	Description
15 – 4	Reserved and must be set to zero.
3	If segment is shared, it can be decreased in size by DosReallocSeg.
2	Segment may be discarded by the system in low memory situations.
1	Segment is shareable through DosGetSeg.
0	Segment is shareable through DosGiveSeg.

**rc** (*USHORT*) – return

Return code descriptions are:

0	NO_ERROR
8	ERROR_NOT_ENOUGH_MEMORY
87	ERROR_INVALID_PARAMETER

## Remarks

DosAllocSeg allows a process to allocate a data segment up to 64KB in size, which is movable and swappable by the system. If your application needs to accommodate a large data structure that exceeds the 64KB limit, DosAllocHuge may be issued to allocate multiple segments as one huge block of memory.

A segment allocated by DosAllocSeg with AllocFlags bit 2 set can be discarded by the system to remedy a low memory situation when the segment is not in use. Upon allocation, a discardable segment is locked and ready for access. The caller issues DosUnlockSeg when it is finished using the segment. The next time the caller needs to access the segment, it must issue DosLockSeg. During the time a segment is locked, it cannot be discarded, but it can still be swapped.

Allocate memory as discardable when it is needed to hold data for only short periods of time; for example, saved bit map images for obscured windows. Once the system discards a segment, the caller must reallocate the segment with DosReallocSeg and regenerate the data. Reallocating the segment automatically locks it for the first access.

A segment may also be designated as shared with another process. If a process issues DosAllocSeg with AllocFlags bits 0 and 1 set, the process can then issue DosGiveSeg to obtain a selector for the sharer to use. The process then passes the selector to the sharer using some means of interprocess communication. For the sharer to access the shared segment, it must issue DosGetSeg with the passed selector. If the shared segment has been designated discardable (AllocFlags bit 2 is also set), the sharer must also issue DosLockSeg to lock the segment.

Memory allocated with DosAllocSeg is freed by a call to DosFreeSeg.

**Note:** This request may be issued from privilege level 2. However, the segment is allocated as a privilege level 3 segment.

# DosAllocSeg — Allocate Segment

FAPI

## Family API Considerations

Some options operate differently in the DOS mode than in the OS/2 mode. Therefore, the following restrictions apply to DosAllocSeg when coding for the DOS mode:

- Requested Size value is rounded up to the next paragraph (16-byte).
- Selector is the actual segment address allocated.
- AllocFlags must be set to zero.

## C Language

```
#define INCL_DOSMEMMGR

USHORT rc = DosAllocSeg(Size, Selector, AllocFlags);

USHORT      Size;          /* Number of bytes requested */
PSEL        Selector;      /* Selector allocated (returned) */
USHORT      AllocFlags;    /* Allocation flags */

USHORT      rc;            /* return code */
```

## Assembler Language

```
EXTRN DosAllocSeg:FAR
INCL_DOSMEMMGR EQU 1

PUSH WORD Size ;Number of bytes requested
PUSH@ WORD Selector ;Selector allocated (returned)
PUSH WORD AllocFlags ;Allocation flags
CALL DosAllocSeg
```

Returns WORD

## Example

This example requests a segment of memory with 30,128 bytes. The segment can be shared with a DosGetSeg API call.

```
#define INCL_DOSMEMMGR

#define NUMBER_OF_BYTES 30128
#define ALLOC_FLAG SEG_GETTABLE

SEL Selector;
USHORT rc;

rc = DosAllocSeg(NUMBER_OF_BYTES, /* # of bytes requested */
                 &Selector,      /* Selector allocated */
                 ALLOC_FLAG);    /* Allocation flags */
```

The following example requests a segment of memory with 4,000 bytes. The following example also shows how to suspend and resume execution of a thread within a process. The main thread creates Thread2 and allows it to begin executing. Thread2 iterates through a loop that prints a line and then sleeps, relinquishing its time slice to the main thread. After one iteration by Thread2, the main thread suspends Thread2 and then resumes it. Subsequently, Thread2 completes the remaining three iterations.

```

#define INCL_DOSPROCESS

#include <os2.h>

#define SEGSIZE      4096    /* Number of bytes requested in segment */
#define ALLOCFLAGS    0      /* Segment allocation flags - no sharing */
#define SLEEPSHORT    5L     /* Sleep interval - 5 milliseconds */
#define SLEEPLONG     75L    /* Sleep interval - 75 milliseconds */
#define RETURN_CODE    0     /* Return code for DosExit() */

VOID APIENTRY Thread2()
{
    USHORT    i;

    /* Loop with four iterations */
    for(i=1; i<5; i++)
    {
        printf("In Thread2, i is now %d\n", i);
        /* Sleep to relinquish time slice to main thread */
        DosSleep(SLEEPSHORT);          /* Sleep interval */
    }
    DosExit(EXIT_THREAD,               /* Action code - end a thread */
            RETURN_CODE);              /* Return code */
}

main()
{
    TID        ThreadID;           /* Thread identification */
    SEL        ThreadStackSel;     /* Segment selector for thread stack */
    PBYTE      StackEnd;           /* Ptr. to end of thread stack */
    USHORT     rc;

    /** Allocate segment for thread stack; make pointer to end of stack. **/
    /** We must allocate a segment in order to preserve segment protection **/
    /** for the thread. **/

    rc = DosAllocSeg(SEGSIZE,        /* Number of bytes requested */
                    &ThreadStackSel, /* Segment selector (returned) */
                    ALLOCFLAGS);     /* Allocation flags - no sharing */
    StackEnd = MAKEP(ThreadStackSel, SEGSIZE-1);

    /** Start Thread2 **/
    if(!rc=DosCreateThread((PFNTHREAD) Thread2, /* Thread address */
                          &ThreadID,           /* Thread ID (returned) */
                          StackEnd))) /* End of thread stack */
        printf("Thread2 created.\n");

    /* Sleep to relinquish time slice to Thread2 */
    if(!rc=DosSleep(SLEEPSHORT)) /* Sleep interval */
        printf("Slept a little to let Thread2 execute.\n");

    /****** Suspend Thread2, do some work, then resume Thread2 *****/
    if(!rc=DosSuspendThread(ThreadID)) /* Thread ID */
        printf("Thread2 SUSPENDED.\n");
    printf("Perform work that will not be interrupted by Thread2.\n");
    if(!rc=DosResumeThread(ThreadID)) /* Thread ID */
        printf("Thread2 RESUMED.\n");
    printf("Now we may be interrupted by Thread2.\n");

    /* Sleep to allow Thread2 to complete */
    DosSleep(SLEEPLONG);          /* Sleep interval */
}

```

# DosAllocShrSeg — Allocate Named Shared Segment

---

This call allocates a named shared memory segment to a process.

**DosAllocShrSeg (Size, Name, Selector)**

## Parameters

**Size (USHORT)** — input

Number of bytes requested. The value specified must be less than or equal to 65535. A value of 0 indicates 65536 bytes.

**Name (PSZ)** — input

Address of the name string associated with the shared memory segment to be allocated. Name specifies the symbolic name for the shared memory segment. The name string is an ASCII string in the format of an OS/2 file name in a subdirectory called \SHAREMEM\. The name string must include the prefix \SHAREMEM\. For example \SHAREMEM\name.

**Selector (PSEL)** — output

Address where the selector of the allocated segment is returned.

**rc (USHORT)** — return

Return code descriptions are:

0	NO_ERROR
8	ERROR_NOT_ENOUGH_MEMORY
123	ERROR_INVALID_NAME
183	ERROR_ALREADY_EXISTS

## Remarks

DosAllocShrSeg allocates a named segment of up to 64KB in size, which is movable and swappable. The segment can be shared by any process that knows the name of the segment.

To access the shared segment, another process issues DosGetShrSeg, specifying the segment name. The selector returned by DosGetShrSeg is the same as the one returned by DosAllocShrSeg.

The maximum number of segments a process can define with DosAllocShrSeg or access with DosGetShrSeg is 256.

**Note:** This request may be issued from privilege level 2. However, the segment is allocated as a privilege level 3 segment.

## C Language

```
#define INCL_DOSMEMMGR
```

```
USHORT rc = DosAllocShrSeg(Size, Name, Selector);
```

USHORT	Size;	/* Number of bytes requested */
PSZ	Name;	/* Name string */
PSEL	Selector;	/* Selector allocated (returned) */
USHORT	rc;	/* return code */

# DosAllocShrSeg – Allocate Named Shared Segment

## Assembler Language

```
EXTRN DosAllocShrSeg:FAR
INCL_DOSMEMMGR      EQU 1

PUSH  WORD    Size           ;Number of bytes requested
PUSH@ ASCIIZ  Name          ;Name string
PUSH@ WORD    Selector       ;Selector allocated (returned)
CALL  DosAllocShrSeg
```

Returns WORD

## Example

This example requests a segment of memory with 27 bytes named "stock.dat".

```
#define INCL_DOSMEMMGR

#define NUMBER_OF_BYTES 27
#define NAME_SEG "\\SHAREMEM\\stock.dat"

SEL  Selector;
USHORT rc;

rc = DosAllocShrSeg(NUMBER_OF_BYTES, /* # of bytes requested */
                    NAME_SEG,        /* Name string */
                    &Selector);      /* Selector allocated */
```



# DosBeep — Generate Sound From Speaker

FAP1

---

This call generates sound from the speaker.

**DosBeep (Frequency, Duration)**

## Parameters

**Frequency** (*USHORT*) — input

Tone in Hertz (cycles per second) in the range 37 through 32767.

**Duration** (*USHORT*) — input

Length of the sound in milliseconds.

**rc** (*USHORT*) — return

Return code descriptions are:

0	NO_ERROR
395	ERROR_INVALID_FREQUENCY

## Remarks

DosBeep executes synchronously. An application program that invokes DosBeep waits until the specified number of milliseconds expire before it resumes execution.

## C Language

```
#define INCL_DOSPROCESS

USHORT rc = DosBeep(Frequency, Duration);

USHORT      Frequency;    /* Hertz (Hz) */
USHORT      Duration;     /* Length of sound */

USHORT      rc;           /* return code */
```

## Assembler Language

```
EXTRN DosBeep:FAR
INCL_DOSPROCESS EQU 1

PUSH WORD Frequency ;Frequency (in Hertz)
PUSH WORD Duration  ;Length of sound (in milliseconds)
CALL DosBeep
```

Returns WORD

## Example

This example generates a beep for 1 second (1,000 milliseconds) at a frequency of 1,380.

```
#define INCL_DOSPROCESS

#define BEEP_FREQUENCY 1380
#define BEEP_DURATION 1000

USHORT rc;

rc = DosBeep(BEEP_FREQUENCY,
             BEEP_DURATION);
```

This call flushes a requesting process's cache buffers for a specific file handle or for all file handles attached to that process. This call is also used with the handle of a named pipe to synchronize a dialog between communicating processes.

**DosBufReset (FileHandle)**

## Parameters

**FileHandle (HFILE)** — input

File handle whose buffers are to be flushed. If FileHandle = 0xFFFFH, all of the process's file handles are flushed.

**rc (USHORT)** — return

Return code descriptions are:

<b>0</b>	<b>NO_ERROR</b>
<b>2</b>	<b>ERROR_FILE_NOT_FOUND</b>
<b>5</b>	<b>ERROR_ACCESS_DENIED</b>
<b>6</b>	<b>ERROR_INVALID_HANDLE</b>

## Remarks

Upon issuing DosBufReset for a file handle, the file's buffers are flushed to disk and its directory entry updated as if the file had been closed; however the file remains in an open state.

Usage of this call to write out all files belonging to the requesting process should be administered with caution. When the files reside on removable media (diskettes), a call to DosBufReset could have the undesirable effect of requiring the user to insert and remove a large number of diskettes.

## Named Pipe Considerations

Issuing DosBufReset for a named pipe performs an operation that is analogous to forcing the buffer cache to disk. The request blocks the calling process at one end of the pipe until all data it has written has been read at the other end of the pipe.

## C Language

```
#define INCL_DOSFILEMGR

USHORT rc = DosBufReset(FileHandle);

HFILE      FileHandle;    /* File handle */
USHORT      rc;           /* return code */
```

## Assembler Language

```
EXTRN DosBufReset:FAR
INCL_DOSFILEMGR EQU 1

PUSH WORD FileHandle ;File handle
CALL DosBufReset

Returns WORD
```

# DosBufReset —

## Commit File's Cache Buffers

FAPI

### Example

This example opens a file, writes some data to the file's buffer, then flushes the file's system buffer.

```
#define INCL_DOSFILEMGR

#define OPEN_FILE 0x01
#define CREATE_FILE 0x10
#define FILE_ARCHIVE 0x20
#define FILE_EXISTS OPEN_FILE
#define FILE_NOEXISTS CREATE_FILE
#define DASD_FLAG 0
#define INHERIT 0x80
#define WRITE_THRU 0
#define FAIL_FLAG 0
#define SHARE_FLAG 0x10
#define ACCESS_FLAG 0x02

#define FILE_NAME "test.dat"
#define FILE_SIZE 800L
#define FILE_ATTRIBUTE FILE_ARCHIVE
#define RESERVED 0L

HFILE FileHandle;
USHORT Wrote;
USHORT Action;
PSZ FileData[100];
USHORT rc;

Action = 2;
strcpy(FileData, "Data...");
if(!DosOpen(FILE_NAME, /* File path name */
            &FileHandle, /* File handle */
            &Action, /* Action taken */
            FILE_SIZE, /* File primary allocation */
            FILE_ATTRIBUTE, /* File attribute */
            FILE_EXISTS | FILE_NOEXISTS, /* Open function type */
            DASD_FLAG | INHERIT, /* Open mode of the file */
            WRITE_THRU | FAIL_FLAG |
            SHARE_FLAG | ACCESS_FLAG,
            RESERVED)) /* Reserved (must be zero) */
if(!DosWrite(FileHandle, /* File handle */
             (PVOID) FileData, /* User buffer */
             sizeof(FileData), /* Buffer length */
             &Wrote)) /* Bytes written */
    rc = DosBufReset(FileHandle); /* File handle */
```

# DosCallback — Privilege Level 2 Call to Privilege Level 3 Routine

This call provides a privilege level 2 IOPL segment to call a privilege level 3 application segment.

<b>DosCallback (Ring3Routine)</b>
-----------------------------------

## Parameters

**Ring3Routine (PFN)** — input

Address of the privilege level 3 application routine to be called.

## Remarks

This function allows a routine executing with I/O privilege (at privilege level 2) to call a segment that is executing at privilege level 3 and also to have the target routine execute at privilege level 3; for example, not to allow/require the target routine to be privilege level 2 conforming.

The requested routine is given control at privilege level 3 and when it completes execution and returns, return is made to the privilege level 2 calling routine.

All registers except FLAGS are passed intact across this call return sequence and may be used to pass parameters or data. Any addresses passed from privilege level 2 to privilege level 3 must be based on privilege level 3 selectors only as the privilege level 3 code does not have proper addressability to any privilege level 2 data selectors.

The privilege level 2 stack can not be used to pass data to the privilege level 3 routine.

If DosCallback is used in a nested fashion such that a privilege level 3 routine issues a call to a privilege level 2 routine while performing a "Callback" operation (after being invoked by "Callback" but before having issued a Far RET back to the privilege level 2 code), any subsequent DosCallback requests must complete execution and issue their corresponding Far RETs in last-in-first-out (LIFO) order.

## C Language

```
#define INCL_DOSDEVICES
```

```
VOID DosCallback(Ring3Routine);
```

```
PFN Ring3Routine; /* Address of privilege level 3 routine */
```

## Assembler Language

```
EXTRN DosCallback:FAR
```

```
INCL_DOSDEVICES EQU 1
```

```
PUSH@ OTHER Ring3Routine ;Address of privilege level 3 routine
```

```
CALL DosCallback
```

Returns NONE

# DosCallNmPipe — Perform Procedure Call Transaction

---

This call performs a “procedure call” transaction using a message pipe.

**DosCallNmPipe** (**FileName**, **InBuffer**, **InBufferLen**, **OutBuffer**, **OutBufferLen**, **BytesOut**, **Timeout**)

## Parameters

**FileName** (*PSZ*) — input

Address of the ASCIIZ name of the pipe to be opened. Pipes are named \PIPE\FileName.

**InBuffer** (*PBYTE*) — input

Address of the buffer to write to the pipe.

**InBufferLen** (*USHORT*) — input

Number of bytes to be written.

**OutBuffer** (*PBYTE*) — input/output

Address of the buffer for returned data.

**OutBufferLen** (*USHORT*) — input

Maximum size (number of bytes) of returned data.

**BytesOut** (*PUSHORT*) — output

Address of the variable where the system returns the number of bytes actually read (returned).

**Timeout** (*ULONG*) — input

Maximum time to wait for pipe availability.

**rc** (*USHORT*) — return

Return code descriptions are:

<b>0</b>	<b>NO_ERROR</b>
<b>2</b>	<b>ERROR_FILE_NOT_FOUND</b>
<b>11</b>	<b>ERROR_BAD_FORMAT</b>
<b>230</b>	<b>ERROR_BAD_PIPE</b>
<b>231</b>	<b>ERROR_PIPE_BUSY</b>
<b>233</b>	<b>ERROR_PIPE_NOT_CONNECTED</b>
<b>234</b>	<b>ERROR_MORE_DATA</b>

## Remarks

This call is intended for use only on duplex message pipes. If this call is issued for a pipe that is not a duplex message pipe, **ERROR\_BAD\_FORMAT** is returned.

This call has the combined effect on a named pipe of **DosOpen**, **DosTransactNmPipe**, and **DosClose**. It provides an efficient means of implementing local and remote procedure-call (RPC) interfaces between processes.

## DosCallNmPipe — Perform Procedure Call Transaction

### C Language

```
#define INCL_DOSNMPICES

USHORT rc = DosCallNmPipe(FileName, InBuffer, InBufferLen, OutBuffer,
                          OutBufferLen, BytesOut, Timeout);

PSZ      FileName;      /* Pipe name */
PBYTE    InBuffer;      /* Write buffer address */
USHORT    InBufferLen;   /* Write buffer length */
PBYTE    OutBuffer;     /* Read buffer address */
USHORT    OutBufferLen;  /* Read buffer length */
USHORT    BytesOut;      /* Bytes read (returned) */
ULONG     Timeout;       /* Maximum wait time */

USHORT    rc;            /* return code */
```

### Assembler Language

```
EXTRN DosCallNmPipe:FAR
INCL_DOSNMPICES EQU 1

PUSH@ ASCIIZ FileName      ;Pipe name
PUSH@ OTHER InBuffer       ;Write buffer
PUSH WORD InBufferLen      ;Write buffer length
PUSH@ OTHER OutBuffer      ;Read buffer
PUSH WORD OutBufferLen     ;Read buffer length
PUSH@ WORD BytesOut        ;Bytes read (returned)
PUSH DWORD Timeout         ;Maximum wait time
CALL DosCallNmPipe
```

Returns WORD

# DosCaseMap – Perform Case Mapping

FAP1

---

This call performs case mapping on a string of binary values that represent ASCII characters.

<b>DosCaseMap</b> ( <b>Length</b> , <b>Country</b> , <b>BinaryString</b> )
--

## Parameters

**Length** (*USHORT*) – input

Length of the string of binary values to be case mapped.

**Country** (*PCOUNTRYCODE*) – input/output

Address of the country information structure:

**country** (*USHORT*)

Country code identifier is a binary value of the selected country code. 0 is the default country code.

**codepage** (*USHORT*)

Code page identifier is a binary value of the selected code page. 0 is the code page of the current process.

**BinaryString** (*PCHAR*) – input/output

Address of a string of binary characters to be case mapped. They are case mapped in place so the results appear in BinaryString and the input is destroyed.

**rc** (*USHORT*) – return

Return code descriptions are:

0	NO_ERROR
396	ERROR_NLS_NO_COUNTRY_FILE
397	ERROR_NLS_OPEN_FAILED
398	ERROR_NO_COUNTRY_OR_CODEPAGE
399	ERROR_NLS_TABLE_TRUNCATED

## Remarks

DosCaseMap is mainly used to map a lower case character string to an upper case character string. Unless the user replaces the country information file, DosCaseMap only does the conversion from lower case to upper case.

The case map information is taken from the country information file. See the COUNTRY statement in the *IBM Operating System/2 Version 1.2 Command Reference* for information on how to specify the country information file.

If countrycode is 0, the case mapping is performed using the information for the country specified in the COUNTRY statement in CONFIG.SYS.

If countrycode is not 0, the case mapping is performed using the information for that country.

If the code page identifier is 0, the case mapping is performed using the information for the current process code page. Refer to "DosSetCp – Set Code Page" on page 2-320 and the CHCP command in the *IBM Operating System/2 Version 1.2 Command Reference* for information on setting the process code page. If codepage is not 0, the case mapping is performed using the information for that code page.

The returned country dependent information may be for the default country and current process code page or for a specific country and code page.

## C Language

```
typedef struct _COUNTRYCODE { /* ctryc */
    USHORT country;           /* country code */
    USHORT codepage;          /* code page */
} COUNTRYCODE;

#define INCL_DOSNLS

USHORT rc = DosCaseMap(Length, Structure, BinaryString);

USHORT      Length;          /* Length of string to case map */
PCOUNTRYCODE Structure;      /* Input data structure */
PCHAR       BinaryString;    /* Address of binary string */

USHORT      rc;              /* return code */
```

## Assembler Language

```
COUNTRYCODE struc
    ctryc_country dw ? ;country code
    ctryc_codepage dw ? ;code page
COUNTRYCODE ends

EXTRN DosCaseMap:FAR
INCL_DOSNLS EQU 1

PUSH WORD Length ;Length of string to case map
PUSH@ OTHER Structure ;Input data structure
PUSH@ OTHER BinaryString ;Binary string
CALL DosCaseMap

Returns WORD
```

## Example

This example case maps a string for the default country and code page 850.

```
#define INCL_DOSNLS

#define CURRENT_COUNTRY 0
#define NLS_CODEPAGE 850

COUNTRYCODE Country;
CHAR BinString[30];
USHORT rc;

Country.country = CURRENT_COUNTRY; /* Country code */
Country.codepage = NLS_CODEPAGE; /* Code page */
strcpy(BinString, "Howdy"); /* String to map */
rc = DosCaseMap(sizeof(BinString), /* Length of string */
                &Country, /* Input data structure */
                BinString); /* String */
```



# DosChDir —

## Change Current Directory

FAPI

---

This call defines the current directory for the requesting process.

<b>DosChDir (DirName, Reserved)</b>
-------------------------------------

### Parameters

**DirName (PSZ)** — input

Address of the ASCIIZ directory path name.

**Reserved (ULONG)** — input

Reserved and must be set to zero.

**rc (USHORT)** — return

Return code descriptions are:

0	NO_ERROR
2	ERROR_FILE_NOT_FOUND
3	ERROR_PATH_NOT_FOUND
5	ERROR_ACCESS_DENIED
8	ERROR_NOT_ENOUGH_MEMORY
26	ERROR_NOT_DOS_DISK
87	ERROR_INVALID_PARAMETER
108	ERROR_DRIVE_LOCKED
206	ERROR_FILENAME_EXCED_RANGE

### Remarks

The directory path is not changed if any member of the path does not exist. The current directory changes only for the requesting process.

For FSDs, the case of the current directory is set according to the DirName passed in, not according to the case of the directories on disk. For example, if the directory "c:\bin" is created and DosChDir is called with DirName "c:\bin", the current directory returned by DosQCurDir will be "c:\bin".

Programs running without the NEWFILES bit set are allowed to DosChDir to a non-8.3 filename format directory.

DosQSysInfo must be used by an application to determine the maximum path length supported by OS/2. The returned value should be used to dynamically allocate buffers that are to be used to store paths.

### C Language

```
#define INCL_DOSFILEMGR
```

```
USHORT rc = DosChDir(DirName, Reserved);
```

```
PSZ      DirName;    /* Directory path name */
ULONG    0;          /* Reserved (must be zero) */

USHORT    rc;        /* return code */
```

**Assembler Language**

```
EXTRN DosChDir:FAR
INCL_DOSFILEMGR    EQU 1

PUSH@  ASCIIZ  DirName      ;Directory path name string
PUSH   DWORD   0            ;Reserved (must be zero)
CALL   DosChDir
```

Returns WORD

**Example**

This example changes directories to \os2\system.

```
#define INCL_DOSFILEMGR

#define PATH "\\os2\\system"
#define RESERVED 0L

USHORT rc;

    rc = DosChDir(PATH, RESERVED);
```

# DosChgFilePtr — Move File Read/Write Pointer

FAPI

This call moves the read/write pointer in accordance with the type of move specified.

**DosChgFilePtr (FileHandle, Distance, MoveType, NewPointer)**

## Parameters

**FileHandle** (*HFILE*) — input

Handle returned by a previous DosOpen call.

**Distance** (*LONG*) — input

The offset to move, in bytes.

**MoveType** (*USHORT*) — input

Method of moving. Specifies a location in the file from where Distance to move the read/write pointer starts. Values and their meanings are:

Value	Definition
0	Beginning of the file.
1	Current location of the read/write pointer.
2	End of the file. Use this method to determine a file's size.

**NewPointer** (*PULONG*) — output

Address of the new pointer location.

**rc** (*USHORT*) — return

Return code descriptions are:

0	NO_ERROR
1	ERROR_INVALID_FUNCTION
6	ERROR_INVALID_HANDLE

## Remarks

The read/write pointer in a file is a signed 32-bit number. A negative value moves the pointer backward in the file. A positive value moves the pointer forward. DosChgFilePtr cannot be used to seek to a negative position in the file.

This call may not be used for a character device or pipe.

## C Language

```
#define INCL_DOSFILEMGR
```

```
USHORT rc = DosChgFilePtr(FileHandle, Distance, MoveType, NewPointer);
```

```
HFILE      FileHandle;    /* File handle */
LONG       Distance;      /* Distance to move in bytes */
USHORT     MoveType;      /* Method of moving (0, 1, 2) */
PULONG     NewPointer;    /* New Pointer Location */

USHORT     rc;            /* return code */
```

## Assembler Language

```

EXTRN DosChgFilePtr:FAR
INCL_DOSFILEMGR      EQU 1

PUSH  WORD    FileHandle    ;File handle
PUSH  DWORD   Distance     ;Distance to move in bytes
PUSH  WORD    MoveType      ;Method of moving (0, 1, 2)
PUSH@ DWORD   NewPointer    ;New Pointer Location (returned)
CALL  DosChgFilePtr

```

Returns WORD

## Example

This example opens file test.dat, writes some data, and resets the file pointer to the beginning of the file.

```

#define INCL_DOSFILEMGR

#define OPEN_FILE 0x01
#define CREATE_FILE 0x10
#define FILE_ARCHIVE 0x20
#define FILE_EXISTS OPEN_FILE
#define FILE_NOEXISTS CREATE_FILE
#define DASD_FLAG 0
#define INHERIT 0x80
#define WRITE_THRU 0
#define FAIL_FLAG 0
#define SHARE_FLAG 0x10
#define ACCESS_FLAG 0x02

#define FILE_NAME "test.dat"
#define FILE_SIZE 800L
#define FILE_ATTRIBUTE FILE_ARCHIVE
#define RESERVED 0L

HFILE FileHandle;
USHORT Wrote;
USHORT Action;
PUSHORT Local
PSZ FileData[100];
USHORT rc;

Action = 2;
strcpy(FileData, "Data...");
if(!DosOpen(FILE_NAME,          /* File path name */
            &FileHandle,        /* File handle */
            &Action,            /* Action taken */
            FILE_SIZE,          /* File primary allocation */
            FILE_ATTRIBUTE,      /* File attribute */
            FILE_EXISTS | FILE_NOEXISTS, /* Open function type */
            DASD_FLAG | INHERIT | /* Open mode of the file */
            WRITE_THRU | FAIL_FLAG |
            SHARE_FLAG | ACCESS_FLAG,
            RESERVED))          /* Reserved (must be zero) */
if(!DosWrite(FileHandle,        /* File handle */
             (PVOID) FileData,   /* User buffer */
             sizeof(FileData),  /* Buffer length */
             &Wrote))           /* Bytes written */
rc = DosChgFilePtr(FileHandle,  /* File handle */
                   MOVE_DIST,    /* Distance to move in bytes */
                   FILE_BEG,     /* Method of moving */
                   &Local);      /* New pointer location */

```

## DosCLIAccess — Request CLI/STI Privilege

---

This call requests I/O privilege for disabling and enabling interrupts. Access to ports must be granted with DosPortAccess.

**DosCLIAccess ( )**

### Parameters

**rc** (*USHORT*) — return

Return code descriptions are:

0            NO\_ERROR

### Remarks

Applications that only use CLI/STI in IOPL segments must request CLI/STI privilege from the operating system.

Applications that use IN/OUT instructions to I/O ports must request I/O privilege with DosPortAccess. (See "DosPortAccess — Request Port Access" on page 2-233 for more detail). Request for port access also grants CLI/STI privilege from the operating system.

### C Language

```
#define INCL_DOSDEVICES
```

```
USHORT rc = DosCLIAccess(VOID);
```

```
USHORT rc;            /* return code */
```

### Assembler Language

```
EXTRN DosCLIAccess:FAR  
INCL_DOSDEVICES EQU 1
```

```
CALL DosCLIAccess
```

Returns WORD

### Example

This example requests I/O privilege for disabling and enabling interrupts.

```
#define INCL_DOSDEVICES
```

```
USHORT rc;
```

```
rc = DosCLIAccess();    /* Request I/O privilege */
```

This call closes a handle to a file, pipe, or device.

**DosClose (FileHandle)**

## Parameters

**FileHandle** (*HFILE*) — input

Handle returned by a previous `DosOpen`, `DosMakeNmPipe`, or `DosMakePipe` call.

**rc** (*USHORT*) — return

Return code descriptions are:

<b>0</b>	<b>NO_ERROR</b>
<b>2</b>	<b>ERROR_FILE_NOT_FOUND</b>
<b>5</b>	<b>ERROR_ACCESS_DENIED</b>
<b>6</b>	<b>ERROR_INVALID_HANDLE</b>

## Remarks

Issuing `DosClose` with the handle to a file closes a handle to a file, pipe, or device.

If one or more additional handles to a file have been created with `DosDupHandle`, the directory is not updated and all internal buffers are not written to the medium until `DosClose` has been issued for the duplicated handles.

Closing a handle to a device causes the device to be notified of the close, if appropriate.

## Named Pipe Considerations

`DosClose` closes a named pipe by handle. When all handles referencing one end of a pipe are closed, the pipe is considered broken.

If the client end closes, no other process can re-open the pipe until the serving end issues a `DosDisconnectNmPipe` followed by a `DosConnectNmPipe`.

If the server end closes when the pipe is already broken, it is deallocated immediately; otherwise, the pipe is not deallocated until the last client handle is closed.

## C Language

```
#define INCL_DOSFILEMGR

USHORT rc = DosClose(FileHandle);

HFILE      FileHandle;    /* File handle */
USHORT      rc;           /* return code */
```

## Assembler Language

```
EXTRN DosClose:FAR
INCL_DOSFILEMGR EQU 1

PUSH WORD FileHandle ;File handle
CALL DosClose
```

Returns WORD

# DosClose —

## Close File Handle

FAPI

### Example

This example opens a file, then closes it.

```
#define INCL_DOSFILEMGR

#define OPEN_FILE 0x01
#define CREATE_FILE 0x10
#define FILE_ARCHIVE 0x20
#define FILE_EXISTS OPEN_FILE
#define FILE_NOEXISTS CREATE_FILE
#define DASD_FLAG 0
#define INHERIT 0x80
#define WRITE_THRU 0
#define FAIL_FLAG 0
#define SHARE_FLAG 0x10
#define ACCESS_FLAG 0x02

#define FILE_NAME "test.dat"
#define FILE_SIZE 800L
#define FILE_ATTRIBUTE FILE_ARCHIVE
#define RESERVED 0L

HFILE  FileHandle;
USHORT Wrote;
USHORT Action;
PSZ    FileData[100];
USHORT rc;

Action = 2;
strcpy(FileData, "Data...");
if(!DosOpen(FILE_NAME,                /* File path name */
            &FileHandle,              /* File handle */
            &Action,                  /* Action taken */
            FILE_SIZE,                /* File primary allocation */
            FILE_ATTRIBUTE,           /* File attribute */
            FILE_EXISTS | FILE_NOEXISTS, /* Open function type */
            DASD_FLAG | INHERIT |     /* Open mode of the file */
            WRITE_THRU | FAIL_FLAG |
            SHARE_FLAG | ACCESS_FLAG,
            RESERVED))                /* Reserved (must be zero) */
    rc = DosClose(FileHandle);        /* File Handle */
```

This call closes the queue in use by the requesting process.

**DosCloseQueue (QueueHandle)**

## Parameters

**QueueHandle** (*HQUEUE*) – input

Handle returned from a previous DosCreateQueue or DosOpenQueue call.

**rc** (*USHORT*) – return

Return code descriptions are:

0	NO_ERROR
337	ERROR_QUEUE_INVALID_HANDLE

## Remarks

DosCloseQueue is used to terminate further processing of a queue by the requesting process. The actions taken depend on whether the requestor is the owner or a writer of the queue. For all processes, an access count representing all DosOpenQueue calls performed is decremented. For non-owning processes, access is terminated when this count goes to zero. For owning processes, the queue (and its elements) are purged if the access count previously equaled zero. Other processes that have the queue open receive the ERROR\_QUEUE\_INVALID\_HANDLE return code on their next request.

## C Language

```
#define INCL_DOSQUEUES

USHORT rc = DosCloseQueue(QueueHandle);

HQUEUE QueueHandle; /* Handle of queue */

USHORT rc; /* return code */
```

## Assembler Language

```
EXTRN DosCloseQueue:FAR
INCL_DOSQUEUES EQU 1

PUSH WORD QueueHandle ;Queue handle
CALL DosCloseQueue
```

Returns WORD

## Example

This example opens a queue named special.que, then closes it.

```
#define INCL_DOSQUEUES

#define QUE_FIFO 0
#define QUE_NAME "\\QUEUES\\special.que"

HQUEUE QueueHandle;
USHORT rc;

if(!DosCreateQueue(&QueueHandle, /* Queue handle */
                  QUE_FIFO,      /* Ordering to use for elements */
                  QUE_NAME))     /* Queue name string */
    rc = DosCloseQueue(QueueHandle); /* Queue handle */
```



# DosCloseSem — Close System Semaphore

This call closes a handle to a system semaphore, obtained with a DosCreateSem or DosOpenSem request.

**DosCloseSem (SemHandle)**

## Parameters

**SemHandle (HSEM)** — input

Handle returned from a previous DosCreateSem or DosOpenSem call.

**rc (USHORT)** — return

Return code descriptions are:

0	NO_ERROR
102	ERROR_SEM_IS_SET

## Remarks

DosCloseSem is issued to close a handle to a system semaphore. When all processes that obtained handles to the semaphore with DosCreateSem or DosOpenSem requests have issued DosCloseSem, the semaphore is deleted and must be redefined by the next user with a call to DosCreateSem.

If a process has created a nonexclusive system semaphore and terminates while the semaphore is open, the system closes the handle to the semaphore that was returned by DosCreateSem. If the process is currently holding the semaphore when it terminates, OS/2 clears the semaphore and returns ERROR\_SEM\_OWNER\_DIED to the next thread that gets the resource because of a DosSemRequest call. This error message alerts the holder of the semaphore that the semaphore owner may have ended abnormally; consequently, the resource is in an indeterminate state. The thread should take appropriate steps to protect the integrity of the resource. Upon completion of cleanup activity, the thread can release the semaphore by calling DosSemClear. This call resets the error condition flagged in the semaphore data structure and makes the semaphore available to the next user of the resource. The semaphore remains available to all processes that obtained handles to it with DosOpenSem requests until they call DosCloseSem to release the semaphore handles.

If a process has created an exclusive system semaphore and terminates while the semaphore is open, ownership of the semaphore is transferred to the thread executing any exit list routines. If no exit list routines have been identified to the system with DosExitList, the system closes the handle to the semaphore.

## C Language

```
#define INCL_DOSSEMAPHORES

USHORT rc = DosCloseSem(SemHandle);

HSEM      SemHandle;    /* Semaphore handle */

USHORT    rc;           /* return code */
```

## Assembler Language

```
EXTRN DosCloseSem:FAR
INCL_DOSSEMAPHORES EQU 1

PUSH  DWORD SemHandle    ;Semaphore handle
CALL  DosCloseSem

Returns WORD
```

## DosCloseSem — Close System Semaphore

### Example

This example creates a semaphore named `timeout.sem`, then closes it.

```
#define INCL_DOSSEMAPHORES

#define SEM_OWNERSHIP 0
#define SEM_NAME "\\SEM\\timeout.sem"

HSEM SemHandle;
USHORT rc;

if(!DosCreateSem(SEM_OWNERSHIP, /* Indicate ownership */
                &SemHandle, /* Semaphore handle */
                SEM_NAME)) /* Semaphore name string */
    rc = DosCloseSem(SemHandle); /* Semaphore handle */
```

The following example illustrates the serialization of access to a shared resource between threads of the same process. The program creates a nonexclusive system semaphore named `resource.sem`, requests access to the semaphore, clears it, and finally closes the semaphore. For an illustration of notification of events, see the example given in `DosOpenSem`, `DosSemSet`, or `DosSemWait`.

```
#define INCL_DOSSEMAPHORES

#include <os2.h>

#define SEM_NAME "\\SEM\\resource.sem" /* Semaphore name */
#define TIMEOUT 1500L /* Timeout (in milliseconds) */

main()
{
    HSEM SemHandle;
    USHORT rc;

    /* Note: the semaphore could have been created by another */
    /* thread. */
    DosCreateSem(CSEM_PUBLIC, /* Ownership - nonexclusive */
                &SemHandle, /* Semaphore handle (returned) */
                SEM_NAME); /* Semaphore name */
    if(!(rc = DosSemRequest(SemHandle, /* Semaphore Handle */
                          TIMEOUT))) /* Timeout Period */
    {
        /* Semaphore obtained; resource may now be used. */
        /* Clear the semaphore after using resource. */
        if(DosSemClear(SemHandle))
        {
            /* Semaphore exclusively owned by another process -- */
            /* cannot clear now. */
        }
    }
    else
    {
        /* Semaphore not obtained: error processing (i.e. switch on rc) */
    }
    /* Semaphore no longer needed; close it */
    if(DosCloseSem(SemHandle))
    {
        /* Semaphore is still set -- cannot close now */
    }
}
```

# DosConnectNmPipe — Connect Named Pipe

---

This call is issued by the server process and enables the named pipe to be opened by a client.

<b>DosConnectNmPipe (Handle)</b>
----------------------------------

## Parameters

**Handle (HPIPE)** — input

Handle of the named pipe that is returned by DosMakeNmPipe.

**rc (USHORT)** — return

Return code descriptions are:

<b>0</b>	<b>NO_ERROR</b>
<b>95</b>	<b>ERROR_INTERRUPT</b>
<b>109</b>	<b>ERROR_BROKEN_PIPE</b>
<b>230</b>	<b>ERROR_BAD_PIPE</b>
<b>233</b>	<b>ERROR_PIPE_NOT_CONNECTED</b>

## Remarks

The server process, which creates the named pipe with DosMakeNmPipe, prepares the pipe so that it can accept a DosOpen from a client. To prepare the pipe for its first client, the server issues DosConnectNmPipe. To prepare the pipe for the next client, the server issues DosDisconnectNmPipe followed by DosConnectNmPipe.

When DosConnectNmPipe returns, the pipe is in a listening state. A DosOpen to a pipe that is not in a listening state fails. A client can determine the pipe's state by issuing DosPeekNmPipe.

If the client end of the pipe is currently open, DosConnectNmPipe returns immediately and has no effect. If the client end is not open, DosConnectNmPipe either waits until it is open (if blocking mode is set) or else returns immediately with ERROR\_PIPE\_NOT\_CONNECTED (if non-blocking mode is set). In the case where ERROR\_PIPE\_NOT\_CONNECTED is returned, the pipe enters a listening state, permitting a client to issue a successful DosOpen.

If the pipe has been closed by a previous client but is not disconnected by the server, DosConnectNmPipe always returns ERROR\_BROKEN\_PIPE. Multiple DosConnectNmPipe calls can be issued in non-blocking mode; the first one puts the pipe into a listening state (if it is not already open or closing), and subsequent ones simply test the pipe state.

If DosConnectNmPipe is called by the client end of the pipe, ERROR\_BAD\_PIPE is returned. If the wait (in blocking mode only) for the client open is interrupted, the ERROR\_INTERRUPT is returned.

## C Language

```
#define INCL_DOSNMPICES
```

```
USHORT rc = DosConnectNmPipe(Handle);
```

```
HPIPE      Handle;      /* Pipe handle */
```

```
USHORT      rc;          /* return code */
```

## **DosConnectNmPipe — Connect Named Pipe**

### **Assembler Language**

```
EXTRN DosConnectNmPipe:FAR  
INCL_DOSNMPICES EQU 1
```

```
PUSH WORD Handle ;Pipe handle  
CALL DosConnectNmPipe
```

Returns WORD

# DosCopy —

## Copy File

---

This call copies the specified file or subdirectory to the target file or subdirectory.

<b>DosCopy (SourceName, TargetName, OpMode, Reserved)</b>
---

### Parameters

**SourceName (PSZ)** — input

Address of the ASCIIZ path name of the source file, subdirectory, or character device. Global file name characters are not allowed.

**TargetName (PSZ)** — input

Address of the ASCIIZ path name of the target file, subdirectory, or character device. Global file name characters are not allowed.

**OpMode (USHORT)** — input

Word-length bit map that defines how the DosCopy function is done.

Bit	Description
-----	-------------

15–2	Reserved and must be set to zero.
------	-----------------------------------

1	0 = Replace the target file with the source file.
---	---

	1 = Append the source file to the target file's end of data. This is ignored when copying a directory or if the target file doesn't exist.
--	--

0	0 = Do not copy the source file to the target if the file name already exists within the target directory. If a single file is being copied and the target already exists, an error is returned.
---	--

	1 = Copy the source file to the target even if the file name already exists within the target directory.
--	--

**Reserved (ULONG)** — input

Reserved, must be set to zero.

**rc (USHORT)** — return

Return code descriptions are:

0	NO_ERROR
2	ERROR_FILE_NOT_FOUND
3	ERROR_PATH_NOT_FOUND
5	ERROR_ACCESS_DENIED
26	ERROR_NOT_DOS_DISK
32	ERROR_SHARING_VIOLATION
36	ERROR_SHARING_BUFFER_EXCEEDED
87	ERROR_INVALID_PARAMETER
108	ERROR_DRIVE_LOCKED
206	ERROR_FILENAME_EXCED_RANGE
267	ERROR_DIRECTORY

### Remarks

DosCopy copies all files and subdirectories in the source path to the target path. Global file name characters are not allowed in source or target names. The source and the target may be on different drives.

# DosCopy – Copy File

In the event of an I/O error, DosCopy takes the following actions before it terminates:

- If the source name is that of a subdirectory, the file being copied at the time of the error is deleted from the target path.
- If the source name is that of a file to be replaced, the file is deleted from the target path.
- If the source name is that of a file to be appended, the target file is resized to its original size.

Read-only files cannot be replaced by a DosCopy request. If OpMode bit flag0 is set to 1 and read-only files exist in the target, an attempt to replace them with files from the source returns an error.

File attributes are always copied from the source to the target; however extended attributes (EAs) are not copied in every case. DosCopy copies EAs from the source to the target when creating a file or directory or when replacing an existing file on the target. However, it does not copy them when appending an existing file or when copying files to an existing directory on the target.

If a device name is specified as the target, the source name must be a file, not a directory. When the request is issued, OpMode bit flags 0 and 1 are ignored.

DosQSysInfo is called by an application during initialization to determine the maximum path length allowed by OS/2.

## C Language

```
#define INCL_DOSFILEMGR

USHORT rc = DosCopy(SourceName, TargetName, OpMode, 0);

PSZ      SourceName;    /* Source path name */
PSZ      TargetName;    /* Target path name */
USHORT   OpMode;        /* Operation mode */
ULONG    0;             /* Reserved (must be zero) */

USHORT   rc;            /* return code */
```

## Assembler Language

```
EXTRN DosCopy:FAR
INCL_DOSFILEMGR EQU 1

PUSH@ ASCIIZ SourceName ;Source path name string
PUSH@ ASCIIZ TargetName ;Target path name string
PUSH WORD OpMode ;Operation mode
PUSH DWORD 0 ;Reserved (must be zero)
CALL DosCopy

Returns WORD
```

---

This call creates a code segment alias descriptor for a data segment passed as input.

<b>DosCreateCSAlias</b> ( <b>DataSelector</b> , <b>CodeSelector</b> )
---

## Parameters

**DataSelector** (*SEL*) – input  
Data segment selector.

**CodeSelector** (*PSEL*) – output  
Address where the selector of the code segment alias descriptor is returned.

**rc** (*USHORT*) – return  
Return code descriptions are:

0	NO_ERROR
5	ERROR_ACCESS_DENIED

## Remarks

A selector returned by a call to `DosAllocSeg` with no sharing options specified can be used as the data segment specified with `DosCreateCSAlias`. The selector is valid for DS, ES, FS, GS, or SS registers. (The FS and GS registers only exist on the 80386 processor.)

A CS alias segment must be exclusively accessible by the process and cannot be a huge segment. Selectors of shared memory segments and dynamically linked global data segments cannot be used as input for `DosCreateCSAlias`.

The code segment selector returned by `DosCreateCSAlias` is valid for CS. If a procedure is stored in the data segment, it can be called using the CS alias. The procedure may be called from privilege level 3 or I/O privilege level.

Use `DosFreeSeg` to free a CS alias selector created with `DosCreateCSAlias`. Procedures in the segment can continue to be referenced if the data selector for the aliased segment is passed to `DosFreeSeg`, because the CS alias selector is not affected. Once both selectors have been passed to `DosFreeSeg`, the segment is deallocated.

## Family API Considerations

The returned selector is the segment address of the allocated memory. When the returned selector or the original selector is freed, OS/2 immediately deallocates the block of memory.

## C Language

```
#define INCL_DOSMEMMGR

USHORT rc = DosCreateCSAlias(DataSelector, CodeSelector);

SEL      DataSelector; /* Data segment selector */
PSEL     CodeSelector; /* Code segment selector (returned) */

USHORT   rc;          /* return code */
```

**Assembler Language**

```

EXTRN DosCreateCSAlias:FAR
INCL_DOSMEMMGR      EQU 1

PUSH  WORD    DataSelector ;Data segment selector
PUSH0 WORD    CodeSelector ;Code segment selector (returned)
CALL  DosCreateCSAlias

```

Returns WORD

**Example**

This example requests a block of memory (data segment) then requests a descriptor of the segment marking it as a code segment.

```

#define INCL_DOSMEMMGR

#define NUMBER_OF_BYTES 120
#define ALLOC_FLAG SEG_GETTABLE

SEL  CodeSel;
SEL  Selector;
USHORT rc;

if(!DosAllocSeg(NUMBER_OF_BYTES,      /* # of bytes requested */
                &Selector,            /* Selector allocated */
                ALLOC_FLAG))          /* Allocation flags */
    rc = DosCreateCSAlias(Selector,    /* Data segment selector */
                        &CodeSel);    /* Code segment selector */

```



# DosCreateQueue — Create Queue

---

This call creates a queue owned by a creating process.

<b>DosCreateQueue</b> (RWHandle, QueuePrty, QueueName)
--

## Parameters

**RWHandle** (*PHQUEUE*) — output

Address of read/write handle of the queue. The handle is used by the requestor on return.

**QueuePrty** (*USHORT*) — input

Values that indicate the priority ordering algorithm to use for elements placed in the queue.

Value	Definition
0	FIFO queue
1	LIFO queue
2	Priority queue (sender specifies priority zero to 15).

**QueueName** (*PSZ*) — input

Address of the name of the queue. The name string that specifies the name for the queue must include \QUEUES\ as the first element of the path. For example, \QUEUES\RETRIEVE\CONTROL.QUE is a valid queue name. The same name must be specified when calling DosOpenQueue for the process that adds elements to the queue.

**rc** (*USHORT*) — return

Return code descriptions are:

0	NO_ERROR
332	ERROR_QUE_DUPLICATE
334	ERROR_QUE_NO_MEMORY
335	ERROR_QUE_INVALID_NAME
336	ERROR_QUE_INVALID_PRIORITY
337	ERROR_QUE_INVALID_HANDLE

## Remarks

When specifying the name for a queue, the ASCIIZ name string must include the prefix \QUEUES\.

Issuing DosCreateQueue creates a queue that can then be accessed by the creating process. When another process needs to access the queue, it issues DosOpenQueue with the queue's name.

The process that creates the queue owns it and has queue management privileges. Only the owner can peek at the elements in the queue with DosPeekQueue, remove them with DosReadQueue, or purge the queue of all its elements with DosPurgeQueue.

Any process that knows the queue name can open the queue after its creation with DosOpenQueue and place data in it with DosWriteQueue. It can also query the number of elements in the queue with DosQueryQueue and terminate its access to the queue with DosCloseQueue.

Whether a process gains access to the queue by creating or opening it, any thread in that process has access to the queue with equal authority. This provides the capability for multi-server queues.

A queue ceases to exist when its owner issues DosCloseQueue. If other processes use the queue handle for subsequent requests after the owner has closed the queue, ERROR\_QUE\_INVALID\_HANDLE is returned.

# DosCreateQueue — Create Queue

## C Language

```
#define INCL_DOSQUEUES

USHORT rc = DosCreateQueue(RWHandle, QueuePrtY, QueueName);

PHQUEUE    RWHandle;    /* Address to put queue handle (returned) */
USHORT     QueuePrtY;    /* Ordering to use for elements */
PSZ        QueueName;    /* Pointer to queue name string */

USHORT     rc;           /* return code */
```

## Assembler Language

```
EXTRN DosCreateQueue:FAR
INCL_DOSQUEUES EQU 1

PUSH@ WORD    RWHandle    ;Queue handle (returned)
PUSH@ WORD    QueuePrtY    ;Ordering to use for elements
PUSH@ ASCIIZ   QueueName    ;Queue name string
CALL DosCreateQueue

Returns WORD
```

## Example

This example creates a queue named special.que.

```
#define INCL_DOSQUEUES

#define QUE_FIFO 0
#define QUE_NAME "\\QUEUES\\special.que"

HQUEUE QueueHandle;
USHORT rc;

rc = DosCreateQueue(&QueueHandle,    /* Queue handle */
                   QUE_FIFO,         /* Ordering to use for elements */
                   QUE_NAME);        /* Queue name string */
```

# DosCreateSem — Create System Semaphore

---

This call creates a system semaphore used by multiple asynchronous threads to serialize their access to resources.

**DosCreateSem (NoExclusive, SemHandle, SemName)**

## Parameters

**NoExclusive** (*USHORT*) — input

Indicates whether or not the process creating the semaphore wants exclusive use of the semaphore. The meanings for the settings of this flag are:

Value	Definition
0	The creating process has exclusive use of the semaphore. Only threads of the creating process may alter the state of the semaphore with semaphore function calls.
1	The creating process has nonexclusive use of the semaphore. Threads of other processes, as well as those of the creating process, may alter the state of the semaphore with semaphore function calls.

**SemHandle** (*PHYSSEM*) — output

Address of handle of the new system semaphore.

**SemName** (*PSZ*) — input

Address of the name of the system semaphore. A system semaphore is defined within the file system name space as a pseudo file; thus, a semaphore has a full path name. The ASCII string specifying the name must include \SEM\ as the first element of the path. For example, \SEM\RETRIEVE\SIGNAL.SEM is a valid semaphore name.

Although a system semaphore name takes the form of a file in a subdirectory called \SEM, this subdirectory does not exist. System semaphores and their names are kept in memory.

If your application provides long name support for an installable file system, a semaphore name is not restricted to the DOS 8.3 filename format.

**rc** (*USHORT*) — return

Return code descriptions are:

0	NO_ERROR
87	ERROR_INVALID_PARAMETER
100	ERROR_TOO_MANY_SEMAPHORES
123	ERROR_INVALID_NAME
183	ERROR_ALREADY_EXISTS

## Remarks

A call to DosCreateSem creates a system semaphore, which can be manipulated by threads issuing semaphore function calls. Manipulation of a system semaphore is most often used to control access to a serially reusable resource. Manipulation of the semaphore is also used to signal the occurrence of an event to waiting threads.

To control access to a serially reusable resource, a process creates the system semaphore with a call to DosCreateSem. If the users of the resource are asynchronous threads of the creating process, NoExclusive=0 is specified to create an exclusive system semaphore. If the users of the resource include threads of other processes, NoExclusive=1 is specified to create a nonexclusive system semaphore.

DosCreateSem returns a semaphore handle used by the creating process and any of its threads to access the semaphore. If the semaphore is nonexclusive, a process other than the semaphore creator calls DosOpenSem for a handle to access the semaphore.

# DosCreateSem — Create System Semaphore

Ownership of the system semaphore is established by a successful DosSemRequest call. If another thread issues DosSemRequest while the semaphore is owned, the thread can block and wait for the semaphore to become available, or it can return immediately.

After accessing the resource, the owning thread can clear the semaphore by a call to DosSemClear. If the semaphore is an exclusive system semaphore, it has a use count associated with it, which is incremented by DosSemRequest calls and decremented by DosSemClear calls. The semaphore is not actually cleared and made available to the next thread waiting to use the resource until the semaphore has been cleared the same number of times it has been requested. This means that exclusive system semaphores can be used in recursive routines.

If a process has created a nonexclusive system semaphore and terminates while the semaphore is open, the system closes the handle to the semaphore that was returned by DosCreateSem. If the process is currently holding the semaphore when it terminates, OS/2 clears the semaphore and returns ERROR\_SEM\_OWNER\_DIED to the next thread that gets the resource because of a DosSemRequest call. This error message alerts the holder of the semaphore that the semaphore owner may have ended abnormally; consequently, the resource is in an indeterminate state. The thread should take appropriate steps to protect the integrity of the resource. Upon completion of cleanup activity, the thread can release the semaphore by calling DosSemClear. This call resets the error condition flagged in the semaphore data structure and makes the semaphore available to the next user of the resource. The semaphore remains available to all processes that obtained handles to it with DosOpenSem requests until they call DosCloseSem to release the semaphore handles.

If a process has created an exclusive system semaphore and terminates while the semaphore is open, ownership of the semaphore is transferred to the thread executing any exit list routines. If no exit list routines have been identified to the system with DosExitList, the system closes the handle to the semaphore.

In addition to controlling access to serially reusable resources, a nonexclusive system semaphore is also used to signal an event, such as the removal of an element from a queue, to other processes. A process that sets the semaphore waits for another process to clear the semaphore, signaling the event. When the signaling process issues a DosSemClear, any waiting threads resume execution. Calls that support setting and waiting upon a nonexclusive semaphore by one or more threads are DosSemSet, DosSemWait, DosSemSetWait, and DosMuxSemWait.

**Note:** If a thread needs to signal another thread of the same process, a RAM semaphore is used.

## C Language

```
#define INCL_DOSSEMAPHORES

USHORT rc = DosCreateSem(NoExclusive, SemHandle, SemName);

USHORT      NoExclusive; /* Indicate no exclusive ownership */
PHYSSEM     SemHandle;   /* Semaphore handle (returned) */
PSZ         SemName;     /* Semaphore name string */

USHORT      rc;           /* return code */
```

## Assembler Language

```
EXTRN DosCreateSem:FAR
INCL_DOSSEMAPHORES EQU 1

PUSH WORD NoExclusive ;Indicate no exclusive ownership
PUSH@ DWORD SemHandle ;Semaphore handle (returned)
PUSH@ ASCIIZ SemName ;Semaphore name string
CALL DosCreateSem

Returns WORD
```

# DosCreateSem — Create System Semaphore

## Example

This example creates a semaphore named timeout.sem.

```
#define INCL_DOSSEMAPHORES

#define SEM_OWNERSHIP 0
#define SEM_NAME "\\SEM\\timeout.sem"

HSEM SemHandle;
USHORT rc;

rc = DosCreateSem(SEM_OWNERSHIP, /* Indicate ownership */
                  &SemHandle,   /* Semaphore handle */
                  SEM_NAME);     /* Semaphore name string */
```

The following example illustrates the serialization of access to a shared resource between threads of the same process. The program creates a nonexclusive system semaphore named resource.sem, requests access to the semaphore, clears it, and finally closes the semaphore. For an illustration of notification of events, see the example given in DosOpenSem, DosSemSet, or DosSemWait.

```
#define INCL_DOSSEMAPHORES

#include <os2.h>

#define SEM_NAME "\\SEM\\resource.sem" /* Semaphore name */
#define TIMEOUT 1500L                 /* Timeout (in milliseconds) */

main()
{
    HSEM SemHandle;
    USHORT rc;

    /* Note: the semaphore could have been created by another */
    /* thread. */
    DosCreateSem(CSEM_PUBLIC, /* Ownership - nonexclusive */
                &SemHandle,   /* Semaphore handle (returned) */
                SEM_NAME);    /* Semaphore name */
    if(!(rc = DosSemRequest(SemHandle, /* Semaphore Handle */
                           TIMEOUT))) /* Timeout Period */
    {
        /* Semaphore obtained; resource may now be used. */
        /* Clear the semaphore after using resource. */
        if(DosSemClear(SemHandle))
        {
            /* Semaphore exclusively owned by another process -- */
            /* cannot clear now. */
        }
    }
    else
    {
        /* Semaphore not obtained: error processing (i.e. switch on rc) */
    }
    /* Semaphore no longer needed; close it */
    if(DosCloseSem(SemHandle))
    {
        /* Semaphore is still set -- cannot close now */
    }
}
```

# DosCreateThread – Create Another Thread of Execution

---

This call creates an asynchronous thread of execution under the current process.

<b>DosCreateThread</b> ( <i>PgmAddress</i> , <i>ThreadIDWord</i> , <i>NewThreadStack</i> )
--

## Parameters

**PgmAddress** (*PFNTHREAD*) – input

Address within program module where new thread begins execution. This address must not be in an IOPL segment.

**ThreadIDWord** (*PTID*) – output

Address of thread ID of the new thread.

**NewThreadStack** (*PBYTE*) – input

Address of the new thread's stack.

**rc** (*USHORT*) – return

Return code descriptions are:

0	NO_ERROR
8	ERROR_NOT_ENOUGH_MEMORY
89	ERROR_NO_PROC_SLOTS
212	ERROR_LOCKED

## Remarks

OS/2 creates the first thread of a process when it starts the executable file. This thread is dispatched with a regular class priority. To start another thread of execution under the current process, the current thread allocates stack memory and issues `DosCreateThread`. Upon generation of the far call, the thread's initial dispatch point is the address specified for `PgmAddress`. The started thread has a unique stack and register context and the same priority as the requesting thread.

**Note:** The minimum available space on the stack for a thread calling an operating system function must be 4K bytes.

A thread's stack, register context, and priority is the only information maintained by OS/2 that is specific to the thread. The thread shares resources with other threads of the process. Any thread in the process can open a file or device, and any other thread can issue a read or write to that handle. This is also true for pipes, queues, and system semaphores.

The address passed as the `NewThreadStack` value must be the address of the highest byte in the stack. This value is loaded into the SS:PP registers before starting the new thread.

A thread started with `DosCreateThread` terminates upon return of this call or when a `DosExit` is issued. Any thread can temporarily stop the execution of other threads in its process with `DosSuspendThread`, `DosResumeThread`, `DosEnterCritSec`, and `DosExitCritSec` calls. Any thread can also examine and change the priority at which it and other threads execute with `DosGetPrty` and `DosSetPrty`.

**Note:** `DosCreateThread` cannot be issued from within a segment that has I/O privilege (IOPL). If the new thread entry point is in an IOPL code segment, a general protection fault is generated, and the process is terminated.

All code segments execute at a privilege level. Segments for OS/2 applications usually execute at privilege level 3. However, if an application has an IOPL code segment that is executing at privilege level 2 and has to start another thread of execution, `DosCallback` can be issued from the IOPL segment to invoke a privilege level 3 segment. But before the `DosCreateThread` request is made, the IOPL segment's stack must be resized in the privilege level 3 segment by a call to `DosR2StackRealloc`. For more information on IOPL code segments, see *IBM Operating System/2 Version 1.2 I/O Subsystems And Device Support Volume 1*.

# DosCreateThread — Create Another Thread of Execution

## C Language

```
#define INCL_DOSPROCESS

USHORT rc = DosCreateThread(PgmAddress, ThreadIDWord, NewThreadStack);

PFNTHREAD PgmAddress; /* Program address */
PTID ThreadIDWord; /* New thread ID (returned) */
PBYTE NewThreadStack; /* End of stack for new thread */

USHORT rc; /* return code */
```

## Assembler Language

```
EXTRN DosCreateThread:FAR
INCL_DOSPROCESS EQU 1

PUSH DWORD PgmAddress ;Program address
PUSH@ WORD ThreadIDWord ;New thread ID (returned)
PUSH@ OTHER NewThreadStack ;End of stack for new thread
CALL DosCreateThread
```

Returns WORD

## Example

In this example, a second thread is started at TestRoutine with a stack size of 4096 bytes. Remember to compile with Stack checking disabled (-Gs). Also, threads started with DosCreateThread should not use some C library functions. See chapter 6 of the IBM C/2 Language Reference (version 1.1) for a discussion of threads and the C functions \_beginthread and \_endthread. This example can be compiled as follows:

clm -YMS -Gs example.c

```
#define INCL_DOSPROCESS
#define INCL_VIO
#define SLEEP_THREAD1 5000L
#define SLEEP_THREAD2 1000L
#define VIO_HANDLE 0
#define RETURN_CODE 0

TID ThreadID;
BYTE ThreadStackArea[4096];
USHORT rc;

VOID APIENTRY TestRoutine( )
{
    USHORT r;

    r = DosSleep(SLEEP_THREAD2); /* Interval size */
    r = VioWrtTTY("...Thread2...", /* String to be written */
        14, /* Length of string */
        VIO_HANDLE); /* Video handle */
    DosExit(EXIT_THREAD, /* Indicates end thread of process */
        RETURN_CODE); /* Result code */
}

main( )
{
    rc = DosCreateThread( (PFNTHREAD) TestRoutine, /* Program address */
        &ThreadID, /* New thread ID */
        &ThreadStackArea[4095]); /* End of stack for new thread */
    rc = DosSleep(SLEEP_THREAD2); /* Interval size */
    printf("...Thread1...\n");
}
```

The following example shows how to suspend and resume execution of a thread within a process. The main thread creates Thread2 and allows it to begin executing. Thread2 iterates through a loop that prints a line and then sleeps, relinquishing its time slice to the main thread. After one iteration by Thread2, the main thread suspends Thread2 and then resumes it. Subsequently, Thread2 completes the remaining three iterations.

## DosCreateThread — Create Another Thread of Execution

```
#define INCL_DOSPROCESS

#include <os2.h>

#define SEGSIZE      4000  /* Number of bytes requested in segment */
#define ALLOCFLAGS    0    /* Segment allocation flags - no sharing */
#define SLEEPSHORT    5L   /* Sleep interval - 5 milliseconds */
#define SLEEPLONG     75L  /* Sleep interval - 75 milliseconds */
#define RETURN_CODE   0    /* Return code for DosExit() */

VOID APIENTRY Thread2()
{
    USHORT      i;

    /* Loop with four iterations */
    for(i=1; i<5; i++)
    {
        printf("In Thread2, i is now %d\n", i);
        /* Sleep to relinquish time slice to main thread */
        DosSleep(SLEEPSHORT);      /* Sleep interval */
    }
    DosExit(EXIT_THREAD,          /* Action code - end a thread */
            RETURN_CODE);        /* Return code */
}

main()
{
    TID      ThreadID;          /* Thread identification */
    SEL      ThreadStackSel;    /* Segment selector for thread stack */
    PBYTE    StackEnd;          /* Ptr. to end of thread stack */
    USHORT    rc;

    /** Allocate segment for thread stack; make pointer to end of stack. **/
    /** We must allocate a segment in order to preserve segment protection **/
    /** for the thread. **/

    rc = DosAllocSeg(SEGSIZE,          /* Number of bytes requested */
                    &ThreadStackSel,  /* Segment selector (returned) */
                    ALLOCFLAGS);      /* Allocation flags - no sharing */
    StackEnd = MAKEP(ThreadStackSel, SEGSIZE-1);

    /** Start Thread2 **/
    if(!rc=DosCreateThread((PFNTHREAD) Thread2, /* Thread address */
                          &ThreadID,          /* Thread ID (returned) */
                          StackEnd)) /* End of thread stack */
        printf("Thread2 created.\n");

    /* Sleep to relinquish time slice to Thread2 */
    if(!rc=DosSleep(SLEEPSHORT)) /* Sleep interval */
        printf("Slept a little to let Thread2 execute.\n");

    /***** Suspend Thread2, do some work, then resume Thread2 *****/
    if(!rc=DosSuspendThread(ThreadID)) /* Thread ID */
        printf("Thread2 SUSPENDED.\n");
    printf("Perform work that will not be interrupted by Thread2.\n");
    if(!rc=DosResumeThread(ThreadID)) /* Thread ID */
        printf("Thread2 RESUMED.\n");
    printf("Now we may be interrupted by Thread2.\n");

    /* Sleep to allow Thread2 to complete */
    DosSleep(SLEEPLONG);          /* Sleep interval */
}
```



# DosCwait –

## Wait for Child Termination

This call places the current thread in a wait state until an asynchronous child process ends. When the process ends, its process ID and termination code are returned to the caller.

**DosCwait** (**ActionCode**, **WaitOption**, **ReturnCodes**, **ProcessIDWord**, **ProcessID**)

### Parameters

**ActionCode** (*USHORT*) – input

The process whose termination is being waited for.

Value	Definition
0	The child process indicated by ProcessID.
1	The last descendant of the child process indicated by ProcessID.

**WaitOption** (*USHORT*) – input

Return if no child process ends.

Value	Definition
0	Wait if no child process ends or until no child processes are outstanding.
1	Do not wait for child processes to end.

**ReturnCodes** (*PRESULTCODES*) – output

Address of the structure containing the termination code and the result code indicating the reason for the child's termination.

**codeTerminate** (*USHORT*)

The termination code furnished by the system describing why the child terminated.

Value	Definition
0	EXIT (normal)
1	Hard error abort
2	Trap operation
3	Unintercepted DosKillProcess

**codeResult** (*USHORT*)

Result code specified by the terminating process on its last DosExit call.

**ProcessIDWord** (*PPID*) – output

Address of the process ID of the ending process.

**ProcessID** (*PID*) – input

ID of the process whose termination is being waited for:

Value	Definition
0	Any child process.
≠0	The indicated child process and all its descendants.

**rc** (*USHORT*) – return

Return code descriptions are:

0	NO_ERROR
13	ERROR_INVALID_DATA
128	ERROR_WAIT_NO_CHILDREN
129	ERROR_CHILD_NOT_COMPLETE
184	ERROR_NO_CHILD_PROCESS
303	ERROR_INVALID_PROCID

# DosCwait — Wait for Child Termination

## Remarks

DosCwait waits for completion of a child process, whose execution is asynchronous to that of its parent process. The child process is created by a DosExecPgm request with ExecFlags=2 specified. If the child process has multiple threads, the result code returned by DosCwait is the one passed to it by the DosExit request that terminates the process.

DosCwait can also wait for the descendant processes of a child process to complete before it returns. However, it does not report status for descendant processes.

To wait for all child processes and descendant processes to end, issue DosCwait repeatedly, with ActionCode=1 and ProcessID=0 specified, until ERROR\_NO\_CHILD\_PROCESS is returned. The contents of ProcessIDWord can be examined to determine which child the termination codes are from.

If no child processes were started, DosCwait returns with an error. If no child processes terminate, DosCwait can wait until one terminates before returning to the parent, or it can return immediately if it specifies WaitOption=1. This parameter is used to return the result code of a child process that has already ended.

## C Language

```
typedef struct _RESULTCODES { /* resc */

    USHORT codeTerminate;    /* Termination Code */
    USHORT codeResult;       /* Exit Code */

} RESULTCODES;

#define INCL_DOSPROCESS

USHORT rc = DosCwait(ActionCode, WaitOption, ReturnCodes, ProcessIDWord,
                    ProcessID);

USHORT      ActionCode;    /* Execution options */
USHORT      WaitOption;    /* Wait options */
RESULTCODES ReturnCodes;   /* Termination Codes (returned) */
PPID        ProcessIDWord; /* Process ID (returned) */
PID          ProcessID;    /* Process ID of process to wait for */

USHORT      rc;           /* return code */
```

## Assembler Language

```
RESULTCODES struc

    resc_codeTerminate dw ? ;Termination Code
    resc_codeResult    dw ? ;Exit Code

RESULTCODES ends

EXTRN DosCwait:FAR
INCL_DOSPROCESS EQU 1

PUSH WORD ActionCode ;Execution options
PUSH WORD WaitOption ;Wait options
PUSH@ DWORD ReturnCodes ;Termination Codes (returned)
PUSH@ WORD ProcessIDWord ;Process ID (returned)
PUSH WORD ProcessID ;Process ID of process to wait for
CALL DosCwait

Returns WORD
```

## DosCwait — Wait for Child Termination

### Example

This example starts a child session (the program `simple.exe`) and then waits for the child process's termination.

```
#define INCL_DOSPROCESS

#define START_PROGRAM "simple.exe"

CHAR      LoadError[100];
PSZ       Args;
PSZ       Envs;
RESULTCODES ReturnCodes;
USHORT    Pid;
USHORT    rc;

strcpy(Args, "-a2 -l");          /* Pass arguments '-a2' and '-l' */
if(!DosExecPgm(LoadError,        /* Object name buffer */
               sizeof(LoadError), /* Length of object name buffer */
               EXEC_ASYNCRESULT,  /* Asynchronous/Trace flags */
               Args,             /* Argument string */
               Envs,             /* Environment string */
               &ReturnCodes,     /* Termination codes */
               START_PROGRAM))   /* Program file name */
    rc = DosCwait(DCWA_PROCESS, /* Execution options */
                 DCWW_WAIT,     /* Wait options */
                 &ReturnCodes,  /* Termination codes */
                 &Pid,         /* Process ID */
                 ReturnCodes.codeTerminate); /* Process ID of process to wait for */
```

---

This call removes a directory entry associated with a file name.

<b>DosDelete (FileName, Reserved)</b>
---------------------------------------

## Parameters

**FileName (PSZ)** – input

Address of the name of the file to be deleted.

DosQSysInfo is called by an application during initialization to determine the maximum path length allowed by OS/2.

**Reserved (ULONG)** – input

Reserved and must be set to zero.

**rc (USHORT)** – return

Return code descriptions are:

0	NO_ERROR
2	ERROR_FILE_NOT_FOUND
3	ERROR_PATH_NOT_FOUND
5	ERROR_ACCESS_DENIED
26	ERROR_NOT_DOS_DISK
32	ERROR_SHARING_VIOLATION
36	ERROR_SHARING_BUFFER_EXCEEDED
87	ERROR_INVALID_PARAMETER
206	ERROR_FILENAME_EXCED_RANGE

## Remarks

Global file name characters are not permitted.

A file whose read-only attribute is set cannot be deleted. To change the setting of the read-only bit, call DosSetFileMode.

## C Language

```
#define INCL_DOSFILEMGR

USHORT rc = DosDelete(FileName, Reserved);

PSZ      FileName;      /* File name path */
ULONG    0;              /* Reserved (must be zero) */

USHORT    rc;            /* return code */
```

## Assembler Language

```
EXTRN DosDelete:FAR
INCL_DOSFILEMGR EQU 1

PUSH@ ASCIIZ FileName      ;Filename path name string
PUSH  DWORD 0               ;Reserved (must be zero)
CALL  DosDelete
```

Returns WORD

# DosDelete —

## Delete File

FAP1

### Example

This example deletes a file in the current directory named test.dat.

```
#define INCL_DOSFILEMGR

#define FILE_DELETE "test.dat"
#define RESERVED 0L

USHORT rc;

rc = DosDelete(FILE_DELETE, /* File path name */
               RESERVED); /* Reserved (must be zero) */
```

This call gets information about attached devices.

**DosDevConfig (DeviceInfo, Item, Parm)**

## Parameters

**DeviceInfo** (*PVOID*) – output

Address of the byte-wide field containing the requested information.

**Item** (*USHORT*) – input

Device information requested.

Value	Definition
0	Number of printers attached
1	Number of RS232 ports
2	Number of diskette drives
3	Presence of math coprocessor (where 0 = not present, 1 = present)
4	PC Submodel Type ( where the return is the system submodel byte)
5	PC Model Type ( where the return is the system model byte)
6	Display adapter type (where 0 = monochrome mode compatible, 1 = other).

**Parm** (*USHORT*) – input

Reserved for future use and should be set to zero.

**rc** (*USHORT*) – return

Return code descriptions are:

0	NO_ERROR
87	ERROR_INVALID_PARAMETER

## Remarks

The system model (function 5) and submodel (function 4) information is obtained from BIOS.

In addition, the number of devices attached in a PS/2 environment reflect only devices that are "awake". Devices that are "asleep" are not counted.

## C Language

```
#define INCL_DOSDEVICES
```

```
USHORT rc = DosDevConfig(DeviceInfo, Item, Parm);
```

```
PVOID      DeviceInfo; /* Returned information */
USHORT     Item;       /* Item number */
USHORT     Parm;       /* Reserved */

USHORT     rc;         /* return */
```

## Assembler Language

```
EXTRN DosDevConfig:FAR
INCL_DOSDEVICES EQU 1
```

```
PUSH@ OTHER DeviceInfo ;Requested information (returned)
PUSH WORD Item          ;Item number
PUSH WORD Parm          ;Reserved (must be zero)
CALL DosDevConfig
```

Returns WORD

# DosDevConfig —

## Get Device Configuration

FAP1

### Example

This example gets information about model type, monitor and coprocessor and display it.

```
#define INCL_DOSDEVICES

#define MACHINE_MODEL 5
#define DISPLAY_TYPE 6
#define FIND_COPROCESSOR 3
#define RESERVED 0L

BYTE DeviceInfo;
USHORT rc;

if(!DosDevConfig(&DeviceInfo,           /* Returned information */
                 MACHINE_MODEL,         /* Item number */
                 RESERVED))             /* Reserved */
    printf("Model Type %d ",DeviceInfo);

if(!DosDevConfig(&DeviceInfo,           /* Returned information */
                 DISPLAY_TYPE,         /* Item number */
                 RESERVED))             /* Reserved */
    if (DeviceInfo)
        printf("Color display ");
    else
        printf("Mono display ");

if(!DosDevConfig(&DeviceInfo,           /* Returned information */
                 FIND_COPROCESSOR,     /* Item number */
                 RESERVED))             /* Reserved */
    if (DeviceInfo)
        printf("Coprocessor");
    else
        printf("No Coprocessor");
```

This call performs control functions on a device specified by an opened device handle.

**DosDevIOCtl (Data, ParmList, Function, Category, DevHandle)**

## Parameters

**Data (PVOID)** – input  
Address of the data area.

**ParmList (PVOID)** – input  
Address of the command-specific argument list.

**Function (USHORT)** – input  
Device-specific function code.

**Category (USHORT)** – input  
Device category.

**DevHandle (HFILE)** – input  
Device handle returned by DosOpen or a standard (open) device handle.

**rc (USHORT)** – return  
Return code descriptions are:

<b>0</b>	<b>NO_ERROR</b>
<b>1</b>	<b>ERROR_INVALID_FUNCTION</b>
<b>6</b>	<b>ERROR_INVALID_HANDLE</b>
<b>15</b>	<b>ERROR_INVALID_DRIVE</b>
<b>31</b>	<b>ERROR_GEN_FAILURE</b>
<b>87</b>	<b>ERROR_INVALID_PARAMETER</b>
<b>115</b>	<b>ERROR_PROTECTION_VIOLATION</b>
<b>117</b>	<b>ERROR_INVALID_CATEGORY</b>
<b>119</b>	<b>ERROR_BAD_DRIVER_LEVEL</b>
<b>163</b>	<b>ERROR_UNCERTAIN_MEDIA</b>
<b>165</b>	<b>ERROR_MONITORS_NOT_SUPPORTED</b>

## Remarks

Values returned in the range hex FF00 through FFFF are user dependent error codes. Values returned in the range hex FE00 through FEFF are device driver dependent error codes.

Refer to the *IBM Operating System/2 Version 1.2 I/O Subsystems And Device Support Volume 1* for a complete listing of control functions (DevHlp calls).

## Family API Considerations

Some options operate differently in the DOS mode than in the OS/2 mode. Therefore, the following considerations apply to DosDevIOCtl when coding for the DOS mode.

The level of support for DosDevIOCtl is identified by category and function code with a noted restriction if it is not supported by DOS 2.X or DOS 3.X Functions tend to be more restrictive in lower version numbers of DOS.

- Category 1 supported as follows:
  - 41H Set Baud Rate
  - 42H Set Line Control
  - All other category 1 functions are not supported for DOS 2.X and DOS 3.X.
- Category 2 not supported in FAPi
- Category 3 not supported in FAPi



# DosDevIOctl —

## Performs I/O Control for Devices

FAPi

- Category 4 not supported in FAPi
- Category 5 supported in FAPi as follows:
  - 42H Set Frame control - supports IBM Graphics Printers only
  - 44H Set Infinite Retry - for DOS 2.X and DOS 3.X, the function is in effect only for the duration of the calling program
  - 46H Initialize printer
  - 62H Get Frame Control - not supported for DOS 2.X and DOS 3.X
  - 64H Get Infinite Retry
  - 66H Get Printer Status.
- Category 6 not supported in FAPi
- Category 7 not supported in FAPi
- Category 8 supported in FAPi as follows:
  - 00H Lock Drive - not supported for versions below DOS 3.2
  - 01H Unlock Drive - not supported for versions below DOS 3.2
  - 02H Redetermine Media - not supported for versions below DOS 3.2
  - 03H Set Logical Map - not supported for versions below DOS 3.2
  - 20H Block Removable - not supported for versions below DOS 3.2
  - 21H Get Logical Map - not supported for versions below DOS 3.2
  - 43H Set Device Parameters - not supported for DOS 2.X and DOS 3.X
  - 44H Write Track - not supported for DOS 2.X and DOS 3.X
  - 45H Format Track - not supported for DOS 2.X and DOS 3.X
  - 63H Get Device Parameters - not supported for DOS 2.X and DOS 3.X
  - 64H Read Track - not supported for DOS 2.X and DOS 3.X
  - 65H Verify Track - not supported for DOS 2.X and DOS 3.X.
- Category 9 is reserved
- Category 10 (0AH) not supported in FAPi
- Category 11 (0BH) not supported in FAPi.

## C Language

```
#define INCL_DOSDEVICES

USHORT rc = DosDevIOctl(Data, ParmList, Function, Category, DevHandle);

PVOID      Data;          /* Data area */
PVOID      ParmList;      /* Command arguments */
USHORT     Function;      /* Device function */
USHORT     Category;      /* Device category */
HFILE      DevHandle;     /* Specifies the device */

USHORT     rc;            /* return code */
```

## Assembler Language

```
EXTRN DosDevIOctl:FAR
INCL_DOSDEVICES EQU 1

PUSH@ OTHER Data          ;Data area
PUSH@ OTHER ParmList      ;Command arguments
PUSH WORD Function        ;Device function
PUSH WORD Category        ;Device category
PUSH WORD DevHandle       ;Device handle
CALL DosDevIOctl
```

Returns WORD

## DosDevIOCtl2 – Performs I/O Control for Devices

This call performs control functions on a device specified by an opened device handle.

<b>DosDevIOCtl2</b> ( <b>Data</b> , <b>DataLength</b> , <b>ParmList</b> , <b>ParmListLength</b> , <b>Function</b> , <b>Category</b> , <b>DevHandle</b> )
--

### Parameters

**Data** (*PVOID*) – input

Address of the data area.

**DataLength** (*USHORT*) – input

Length of the data buffer.

**ParmList** (*PVOID*) – input

Address of the command-specific argument list.

**ParmListLength** (*USHORT*) – input

Length of the command-specific argument list.

**Function** (*USHORT*) – input

Device-specific function code.

**Category** (*USHORT*) – input

Device category.

**DevHandle** (*HFILE*) – input

Device handle returned by DosOpen or a standard (open) device handle.

**rc** (*USHORT*) – return

Return code descriptions are:

0	NO_ERROR
1	ERROR_INVALID_FUNCTION
6	ERROR_INVALID_HANDLE
15	ERROR_INVALID_DRIVE
31	ERROR_GEN_FAILURE
87	ERROR_INVALID_PARAMETER
115	ERROR_PROTECTION_VIOLATION
117	ERROR_INVALID_CATEGORY
119	ERROR_BAD_DRIVER_LEVEL
163	ERROR_UNCERTAIN_MEDIA
165	ERROR_MONITORS_NOT_SUPPORTED

### Remarks

Values returned in the range hex FF00 through FFFF are user dependent error codes. Values returned in the range hex FE00 through FEFF are device driver dependent error codes.

Refer to the *IBM Operating System/2 Version 1.2 I/O Subsystems And Device Support Volume 1* for a complete listing of control functions (DevHlp calls).

This function provides a generic, expandable IOCTL facility.

A null (zero) value for Data specifies that this parameter is not defined for the generic IOCTL function being specified. A null value for Data causes the value passed in DataLength to be ignored.

A null (zero) value for ParmList specifies that this parameter is not defined for the generic IOCTL function being specified. A null value for ParmList causes the value passed in ParmListLength to be ignored.

The kernel formats a generic IOCTL packet and call the device driver. Since V1.0 and V1.1 device drivers do not understand generic IOCTL packets with DataLength and ParmListLength, the kernel does not pass

## DosDevIOCtl2 — Performs I/O Control for Devices

these fields to the device driver. Device drivers that are marked as being level 2 or higher must support receipt of the generic IOCTL packets with associated length fields.

Do not pass a non-null pointer with a zero length.

### C Language

```
#define INCL_DOSDEVICES
```

```
USHORT rc = DosDevIOCtl2(Data, ParmList, Function, Category, DevHandle);
```

```
PVOID      Data;          /* Data area */
USHORT      DataLength    /* Data area length */
PVOID      ParmList;      /* Command arguments */
USHORT      ParmListLength /* Command arguments list length */
USHORT      Function;      /* Device function */
USHORT      Category;      /* Device category */
HFILE      DevHandle;      /* Specifies the device */

USHORT      rc;            /* return code */
```

### Assembler Language

```
EXTRN DosDevIOCtl2:FAR
INCL_DOSDEVICES EQU 1
```

```
PUSH@ OTHER Data          ;Data area
PUSH WORD DataLength      ;Data area length
PUSH@ OTHER ParmList      ;Command arguments
PUSH WORD ParmListLength  ;Command arguments list length
PUSH WORD Function        ;Device function
PUSH WORD Category        ;Device category
PUSH WORD DevHandle       ;Device handle
CALL DosDevIOCtl2
```

Returns WORD

# DosDisconnectNmPipe — Disconnect Named Pipe

---

This call forces a named pipe to close.

<b>DosDisconnectNmPipe (Handle)</b>
-------------------------------------

## Parameters

**Handle (HPIPE)** — input

Handle of the named pipe that is returned by DosMakeNmPipe.

**rc (USHORT)** — return

Return code descriptions are:

0	NO_ERROR
109	ERROR_BROKEN_PIPE
230	ERROR_BAD_PIPE

## Remarks

The server process of a named pipe issues DosDisconnectNmPipe followed by DosConnectNmPipe to prepare the pipe for the next client.

If the client end of the pipe is open when DosDisconnectNmPipe is issued, it is forced to close, and the client gets an error code on its next operation. Forcing the client end to close may cause data to be discarded that has not yet been read by the client. If the client end is currently closing (DosClose has been issued), DosDisconnectNmPipe acknowledges the close and makes the pipe available to be opened by the next client after a DosConnectNmPipe is issued.

A client that gets forced off a pipe by a DosDisconnectNmPipe must issue DosClose to free the handle resource. Although DosDisconnectNmPipe makes the client's handle invalid, it does not free the client's handle.

Any threads that are blocked on the pipe are awakened by DosDisconnectNmPipe. A thread blocked on the pipe by a DosWrite returns ERROR\_BROKEN\_PIPE. A thread blocked on the pipe by a DosRead returns BytesRead = 0, indicating EOF.

## C Language

```
#define INCL_DOSNMPICES

USHORT rc = DosDisconnectNmPipe(Handle);

HPIPE      Handle;      /* Pipe handle */

USHORT      rc;          /* return code */
```

## Assembler Language

```
EXTRN DosDisconnectNmPipe:FAR
INCL_DOSNMPICES EQU 1

PUSH WORD Handle ;Pipe handle
CALL DosDisconnectNmPipe

Returns WORD
```

# DosDupHandle — Duplicate File Handle

FAP1

This call returns a new file handle for an open file, which refers to the same position in the file as the old file handle.

**DosDupHandle** (OldFileHandle, NewFileHandle)

## Parameters

**OldFileHandle** (*HFILE*) — input  
Current file handle.

**NewFileHandle** (*PHFILE*) — input/output  
Address of a Word. On input, values and their meanings are:

Value	Definition
FFFFH	Allocate a new file handle and return it here.
≠FFFFH	Assign this value as the new file handle. A valid value is any of the handles assigned to standard I/O, or the handle of a file currently opened by the process.

On output, a value of FFFFH returns a value for NewFileHandle, allocated by OS/2.

**rc** (*USHORT*) — return  
Return code descriptions are:

0	NO_ERROR
4	ERROR_TOO_MANY_OPEN_FILES
6	ERROR_INVALID_HANDLE
114	ERROR_INVALID_TARGET_HANDLE

## Remarks

Duplicating the handle duplicates and ties all handle-specific information between OldFileHandle and NewFileHandle. For example, if you move the read/write pointer of either handle by a DosRead, DosWrite, or DosChgFilePtr function call, the pointer for the other handle is also changed.

The valid values for NewFileHandle include the following handles for standard I/O, which are always available to the process:

0000H	Standard input
0001H	Standard output
0002H	Standard error.

If a file handle value of a currently open file is specified in NewFileHandle, the file handle is closed before it is redefined as the duplicate of OldFileHandle. Avoid using arbitrary values for NewFileHandle.

Issuing a DosClose against a file handle does not affect the duplicate handle.

## C Language

```
#define INCL_DOSFILEMGR

USHORT rc = DosDupHandle(OldFileHandle, NewFileHandle);

HFILE      OldFileHandle; /* Existing file handle */
PHFILE     NewFileHandle; /* New file handle (returned) */

USHORT     rc;            /* return code */
```

## Assembler Language

```

EXTRN DosDupHandle:FAR
INCL_DOSFILEMGR EQU 1

PUSH WORD OldFileHandle ;Existing file handle
PUSH@ WORD NewFileHandle ;New file handle (returned)
CALL DosDupHandle

Returns WORD

```

## Example

This example opens a file, creates a second file handle, then closes the file with the second handle.

```

#define INCL_DOSFILEMGR

#define OPEN_FILE 0x01
#define CREATE_FILE 0x10
#define FILE_ARCHIVE 0x20
#define FILE_EXISTS OPEN_FILE
#define FILE_NOEXISTS CREATE_FILE
#define DASD_FLAG 0
#define INHERIT 0x80
#define WRITE_THRU 0
#define FAIL_FLAG 0
#define SHARE_FLAG 0x10
#define ACCESS_FLAG 0x02

#define FILE_NAME "test.dat"
#define FILE_SIZE 800L
#define FILE_ATTRIBUTE FILE_ARCHIVE
#define RESERVED 0L

HFILE FileHandle;
HFILE NewHandle
USHORT Wrote;
USHORT Action;
PSZ FileData[100];
USHORT rc;

Action = 2;
strcpy(FileData, "Data...");
if(!DosOpen(FILE_NAME, /* File path name */
            &FileHandle, /* File handle */
            &Action, /* Action taken */
            FILE_SIZE, /* File primary allocation */
            FILE_ATTRIBUTE, /* File attribute */
            FILE_EXISTS | FILE_NOEXISTS, /* Open function type */
            DASD_FLAG | INHERIT, /* Open mode of the file */
            WRITE_THRU | FAIL_FLAG |
            SHARE_FLAG | ACCESS_FLAG,
            RESERVED)) /* Reserved (must be zero) */
    rc = DosDupHandle(FileHandle, /* Existing file handle */
                    &NewHandle); /* New file handle */

```

## DosEditName — Search for and Edit Names of File Objects

---

This call edits file and subdirectory names indirectly by transforming one ASCII string into another, using global file name characters for editing or search operations on the string.

**DosEditName (EditLevel, SourceString, EditString, TargetBuf, TargetBufLen)**

### Parameters

**EditLevel** (*USHORT*) — input

The level of editing semantics to use in transforming the source string. The value of EditLevel must be 0001H for OS/2 Version 1.2.

**SourceString** (*PSZ*) — input

Address of the ASCIIZ string to transform. Global file name characters are specified only in the subdirectory or file name component of the path name and are interpreted as search characters.

**EditString** (*PSZ*) — input

Address of the ASCIIZ string to use for editing. Global file name characters specified in the edit string are interpreted as editing characters. Because only the name component of a path name is transformed, this string does not include the path component.

**TargetBuf** (*PBYTE*) — output

Address of the buffer to store the resulting ASCIIZ string in.

**TargetBufLen** (*USHORT*) — input

The length of the buffer to store the resulting string in.

**rc** (*USHORT*) — return

Return code descriptions are:

0	NO_ERROR
87	ERROR_INVALID_PARAMETER
123	ERROR_INVALID_NAME

### Remarks

DosEditName is used to search for and edit names of files and subdirectories. This call is typically used in conjunction with calls like DosMove and DosCopy, which do not permit the use of global file name characters, to perform repetitive operations on files.

As an example of an editing operation, a SourceString of "foo.bar" specified with an EditString of "\*.baz" results in "FOO.BAZ" being returned. In the editing process, the string is changed to uppercase.

Global file name characters have two sets of semantics; one for searching and one for editing. If they are specified in SourceString, they are interpreted as search characters. If they are specified in EditString, they are interpreted as editing characters.

Use of the OS/2 COPY utility illustrates this difference in semantics. For example, if a user enters:

```
copy *.old *.new
```

In the source, the "\*" acts as a search character and determines which files to return to the user. In the target, the "\*" functions as an editing character by constructing new names for the matched files.

When used as search characters in SourceString, global file name characters simply match files and behave like any other search characters. They have the following meanings:

- . The "." has no special meaning itself but "?" gives it one.
- \* The "\*" matches 0 or more characters, any character, including a blank. The matching operation does not cross the null character or the backslash (\), which means only the file name is matched, not an entire path.

## DosEditName —

# Search for and Edit Names of File Objects

? The "?" matches 1 character, unless what it would match is a "." or the terminating null characters, in which case it matches 0 characters. It also doesn't cross "\".

Any character other than \* and ? matches itself, including ".".

Searching is case-insensitive.

Any file name that does not have a period (.) in it gets an implicit one automatically appended to the end during searching operations. For example, searching for "foo." would return "foo".

When used as editing characters in EditString, global file name characters have the following meanings:

- . The "." has a special meaning for editing. The "." in the target synchronizes pointers. It causes the source pointer to match a corresponding pointer to the "." in the target. Counting starts from the left of the pointers.
- ? The "?" copies one character, unless what it would copy is a ".", in which case it copies 0. It also copies 0 characters when the end of the source string is reached.
- \* The "\*" copies characters from the source to the target until it finds a source character that matches the character following it in the target.

Editing is case-insensitive and case-preserving. If conflicts arise between the case of the source and editing string, the case of the editing string is used. For example:

```
source string:    "file.txt"
editing string:   "**E.TMP"
destination string: "file.TMP"
```

```
copy file.txt *E.tmp -> file.tmp
```

## C Language

```
#define INCL_DOSFILEMGR
```

```
USHORT rc = DosEditName(EditLevel, SourceString, EditString, TargetBuf, TargetBufLen);
```

```
USHORT      EditLevel;    /* Level of meta editing semantics */
PSZ         SourceString; /* String to transform */
PSZ         EditString;   /* Editing string */
PBYTE       TargetBuf;    /* Destination string buffer */
USHORT      TargetBufLen; /* Destination string buffer length */

USHORT      rc;           /* return code */
```

## Assembler Language

```
EXTRN DosEditName:FAR
INCL_DOSFILEMGR EQU 1
```

```
PUSH WORD EditLevel ;Level of meta editing semantics
PUSH@ ASCIIZ SourceString ;String to transform
PUSH@ ASCIIZ EditString ;Editing string
PUSH@ OTHER TargetBuf ;Destination string buffer (returned)
PUSH WORD TargetBufLen ;Destination string buffer length
CALL DosEditName
```

Returns WORD



# DosEnterCritSec —

## Enter Critical Section of Execution

---

This call disables thread switching for the current process.

**DosEnterCritSec ( )**

### Parameters

**rc** (*USHORT*) — return

Return code descriptions are:

<b>0</b>	<b>NO_ERROR</b>
<b>484</b>	<b>ERROR_CRITSEC_OVERFLOW</b>

### Remarks

DosEnterCritSec causes other threads in the process to block themselves and give up their time slice. After a DosEnterCritSec request is made, no dynamic link calls should be made until the corresponding DosExitCritSec call is completed.

If a signal occurs, thread 1 begins execution to process the signal even though another thread in the process has a DosEnterCritSec active. (Thread 1 of a process is its initial thread of execution, not a thread created with the DosCreateThread call.) Any processing done by thread 1 to satisfy the signal must not include accessing the critical resource intended to be protected by the DosEnterCritSec request.

A count is maintained of the number of times a DosEnterCritSec request is made without a corresponding DosExitCritSec. The count is incremented by DosEnterCritSec and decremented by DosExitCritSec. Normal thread dispatching is not restored until the count is 0. The outstanding DosEnterCritSec count is maintained in a word. If overflow occurs, the count is set to the maximum value, no operation is performed, and the request returns with an error.

A thread can also execute code without having to give up time slices to other threads in its process if it requests a priority class that is higher than those of the other threads. A thread's priority is examined and changed with DosGetPrty and DosSetPrty.

### C Language

```
#define INCL_DOSPROCESS

USHORT rc = DosEnterCritSec(VOID);

USHORT rc; /* return code */
```

### Assembler Language

```
EXTRN DosEnterCritSec:FAR
INCL_DOSPROCESS EQU 1
```

```
CALL DosEnterCritSec
```

Returns WORD

### Example

This example enters a section that will not be pre-empted, performs a simple task, and then exits quickly.

```
#define INCL_DOSPROCESS

USHORT flag;

DosEnterCritSec(); /* Enter critical code section */
flag = TRUE; /* Perform some work */
DosExitCritSec(); /* Exit critical code section */
```

## DosEnumAttribute — Enumerate the extended attributes

This call identifies extended attributes for a specific file or subdirectory.

**DosEnumAttribute** (**RefType**, **FileRef**, **EntryNum**, **EnumBuf**, **EnumBufSize**,  
**EnumCnt**, **InfoLevel**, **Reserved**)

### Parameters

**RefType** (*USHORT*) — input

A value that indicates the contents of FileRef.

Value	Definition
0	Handle of a file.
1	ASCIIZ name of a file or subdirectory.

**FileRef** (*PVOID*) — input

Address of the handle of a file returned by a DosOpen or DosOpen2 request; or the ASCIIZ name of a file or subdirectory.

**EntryNum** (*ULONG*) — input

Ordinal of an entry in the file object's EA list, which indicates where in the list to begin the return of EA information. The value 0 is reserved. A value of 1 indicates the file object's first EA; a value of 2, the second; and so on.

**EnumBuf** (*PVOID*) — output

Address of the buffer where EA information is returned. Level 1 information is returned in the following format:

**Reserved** (*UCHAR*)

Zero.

**cbName** (*UCHAR*)

Length of name excluding NULL.

**cbValue** (*USHORT*)

Length of value.

**szName** (*UCHAR*)

Variable length ascii name.

**EnumBufSize** (*ULONG*) — output

Size of EnumBuf.

**EnumCnt** (*PULONG*) — input/output

Address of, on input, the number of EAs for which information is requested. A value of -1 requests information be returned for as many EAs whose information fits in EnumBuf.

On output, the actual number of EAs for which information is returned. When this value is greater than 1, enumerated information is returned in a packed list. That is, information for the next EA will be stored adjacent to the previous one.

**InfoLevel** (*ULONG*) — input

Level of information required. Only the value 1 can be specified, indicating return of level 1 information.

**Reserved** (*ULONG*) — input

Reserved and must be set to zero.

**rc** (*USHORT*) — return

Return code descriptions are:

0	NO_ERROR
3	ERROR_PATH_NOT_FOUND
5	ERROR_ACCESS_DENIED

## DosEnumAttribute — Enumerate the extended attributes

FAP1

6	ERROR_INVALID_HANDLE
8	ERROR_NOT_ENOUGH_MEMORY
87	ERROR_INVALID_PARAMETER
111	ERROR_BUFFER_OVERFLOW
124	ERROR_INVALID_LEVEL
206	ERROR_FILENAME_EXCED_RANGE

### Remarks

The structure returned by `DosEnumAttribute` is used to calculate the size of the buffer required to hold the full extended attribute (FEA) information for a `DosQPathInfo` or `DosQFileInfo` call that actually gets the FEA. The size of buffer required to hold the FEA information is calculated as follows:

One byte (for `fea_Reserved`) +  
One byte (for `fea_cbName`) +  
Two bytes (for `fea_cbValue`) +  
Value of `cbName` (for the name of the EA) +  
One byte (for terminating NULL in `fea_cbName`) +  
Value of `cbValue` (for the value of the EA)

A process can continue through a file's EA list by reissuing `DosEnumAttribute` with `EntryNum` set to the value specified in the previous call plus the value returned in `EnumCnt`.

`DosEnumAttribute` does not control the specific ordering of EAs; it merely identifies them. Like the files they are associated with, extended attributes can have multiple readers and writers. If a file is open in a sharing mode that allows other processes to modify the file's EA list, calling `DosEnumAttribute` repetitively to back up to an EA's position may return inconsistent results. For example, it is possible for another process to edit the EA list with `DosSetFileInfo` or `DosSetPathInfo` between calls by your process to `DosEnumAttribute`. Thus, the EA returned when `EntryNum` is 11 for the first call may not be the same EA returned when `EntryNum` is 11 for the next call.

To prevent the modification of EAs between calls to `DosEnumAttribute` for a specified file handle or file name, the caller must open the file in deny-write sharing mode before it calls `DosEnumAttribute`. If a subdirectory name is specified, modification by other processes is not a concern, because no sharing is possible.

For `RefType` = 1, the EAs returned are current only when the call was made, and they may have been changed by another thread or process in the meantime.

### C Language

```
typedef struct _DENA1 { /* level 1 info returned from DosEnumAttribute */
    UCHAR reserved; /* 0 */
    UCHAR cbName; /* length of name excluding NULL */
    USHORT cbValue; /* length of value */
    UCHAR szName[1]; /* variable length asciiz name */
} DENA1;

#define INCL_DOSFILEMGR

USHORT rc = DosEnumAttribute(RefType, FileRef, EntryNum, EnumBuf,
                             EnumBufSize, EnumCnt, InfoLevel, Reserved);

USHORT RefType; /* Type of reference */
PVOID FileRef; /* Handle or Name */
ULONG EntryNum; /* Starting entry in EA list */
PVOID EnumBuf; /* Data buffer */
ULONG EnumBufSize; /* Data buffer size */
PULONG EnumCnt; /* Count of entries to return */
ULONG InfoLevel; /* Level of information requested */
ULONG 0; /* Reserved (must be zero) */

USHORT rc; /* return code */
```

## DosEnumAttribute — Enumerate the extended attributes

### Assembler Language

```

DENAL1  struc          ;level 1 info returned from DosEnumAttribute

    level_reserved db      ? ;0
    level_cbName   db      ? ;length of name excluding NULL
    level_cbValue  dw      ? ;length of value
    level_szName   db  1 dup (?) ;variable length asciiz name

DENAL1  ends

EXTRN DosEnumAttribute:FAR
INCL_DOSFILEMGR    EQU 1

PUSH WORD   RefType      ;Type of reference
PUSH@ OTHER FileRef      ;Handle or Name
PUSH DWORD  EntryNum     ;Starting entry in EA list
PUSH@ OTHER EnumBuf      ;Data buffer (returned)
PUSH DWORD  EnumBufSize  ;Data buffer size
PUSH@ DWORD EnumCnt      ;Count of entries to return
PUSH DWORD  InfoLevel    ;Level of information requested
PUSH DWORD  0            ;Reserved (must be zero)
CALL DosEnumAttribute

```

# DosErrClass — Classify Error Codes

FAPI

This call helps OS/2 applications respond to error codes (return codes) received from OS/2.

**DosErrClass** (*Code, Class, Action, Locus*)

## Parameters

**Code** (*USHORT*) — input  
Error code returned by an OS/2 function.

**Class** (*PUSHORT*) — output  
Address of the classification of an error.

**Action** (*PUSHORT*) — output  
Address of the action for an error.

**Locus** (*PUSHORT*) — output  
Address of the origin of an error.

**rc** (*USHORT*) — return  
Return code descriptions are:

0            NO\_ERROR

## Remarks

The input is a return code returned from another function call, and the output is a classification of the return and recommended action. Depending on the application, the recommended action could be followed, or a more specific application recovery could be performed.

The following values are returned in Class, Action, and Locus:

### Class Definitions

Value	Mnemonic	Description
1	OUTRES	Out of resources
2	TEMPSIT	Temporary situation
3	AUTH	Authorization failed
4	INTRN	Internal error
5	HRDFAIL	Device hardware failure
6	SYSFAIL	System failure
7	APPERR	Probable application error
8	NOTFND	Item not located
9	BADFMT	Bad format for call/data
10	LOCKED	Resource/data locked
11	MEDIA	Incorrect media, CRC error
12	ALREADY	Resource/action already taken/done/exists
13	UNK	Unclassified
14	CANT	Can't perform requested action
15	TIME	Timeout

## Action Definitions

Value	Mnemonic	Description
1	RETRY	Retry immediately
2	DLYRET	Delay and retry
3	USER	Bad user input - get new values
4	ABORT	Terminate in an orderly manner
5	PANIC	Terminate immediately
6	IGNORE	Ignore error
7	INTRET	Retry after user intervention

## Locus Definitions

Value	Mnemonic	Description
1	UNK	Unknown
2	DISK	Random access device such as a disk
3	NET	Network
4	SERDEV	Serial device
5	MEM	Memory

## Family API Considerations

Some options operate differently in the DOS mode than in the OS/2 mode. Therefore, the following considerations apply to DosErrClass when coding for the DOS mode:

When DosErrClass is called by a family application, it returns a valid error classification for returns that have occurred. The classifications of a given return code may not be the same for the Family API and the OS/2 mode applications.

## C Language

```
#define INCL_DOSMISC

USHORT rc = DosErrClass(Code, Class, Action, Locus);

USHORT      Code;      /* Error code for analysis */
PUSHORT      Class;     /* Error classification (returned) */
PUSHORT      Action;    /* Recommended action (returned) */
PUSHORT      Locus;     /* Error locus (returned) */

USHORT      rc;         /* return code */
```

## Assembler Language

```
EXTRN DosErrClass:FAR
INCL_DOSMISC EQU 1

PUSH WORD Code      ;Error code for analysis
PUSH@ WORD Class    ;Error classification (returned)
PUSH@ WORD Action    ;Recommended action (returned)
PUSH@ WORD Locus     ;Error locus (returned)
CALL DosErrClass
```

Returns WORD

# DosErrClass — Classify Error Codes

FAP1

## Example

This example attempts to delete a non-existent file. The error returned is then plugged into DosErrClass for more information about the error and what actions should be taken.

```
#define INCL_DOSQUEUES

#define RESERVED 0L
#define FILE_DELETE "adlkjf.dkf"

USHORT Error;
USHORT Class;
USHORT Action;
USHORT Locus;
USHORT rc;

Error = DosDelete(FILE_DELETE, /* File name path */
                  RESERVED); /* Reserved (must be zero) */
rc = DosErrClass(Error, /* Error code for analysis */
                &Class, /* Error classification */
                &Action, /* Recommended action */
                &Locus); /* Error locus */
```

This call allows an OS/2 process to receive hard error notification without generating a hard error signal.

**DosError (Flag)**

## Parameters

**Flags** (*USHORT*) — input

Bit field, defined in the following example (the unused high-order bits are reserved and must be set to zero).

Bit	Description
15 – 2	Reserved, set to zero.
1	0 = Enable exception popups. 1 = Disable exception popups.
0	0 = Disable hard error popups (fail requests). 1 = Enable hard error popups.

**rc** (*USHORT*) — return

Return code descriptions are:

0	NO_ERROR
87	ERROR_INVALID_PARAMETER

## Remarks

DosError allows an OS/2 process to disable user notification if a program (or untrapped numeric processor) exception occurs. If end user notification is disabled, and if one of these exceptions occurs, the process is terminated.

Hard errors generated under a process that has issued a DosError call are failed, and the appropriate error code is returned. The default situation is both hard error pop-ups and exception pop-ups are enabled, if DosError is not issued.

## Family API Considerations

Some options operate differently in the DOS mode than in the OS/2 mode. Therefore, the following restriction applies to DosError when coding for the DOS mode:

For Flag, a value of 0000 causes all subsequent INT 24s to be failed until a subsequent call with a value of 1 is issued.

**Note:** Since INT 24 is not issued in DOS mode, this call has no effect when running in DOS mode.

## C Language

```
#define INCL_DOSMISC

USHORT rc = DosError(Flag);

USHORT      Flags;          /* Action flags */
USHORT      rc;             /* return code */
```



# DosError —

## Enable Hard Error Processing

FAPI

### Assembler Language

```
EXTRN DosError:FAR
INCL_DOSMISC      EQU 1

PUSH  WORD  Flags      ;Action flags
CALL  DosError
```

Returns WORD

### Example

This example disables hard error popups and exception popups, then re-enables them.

```
#define INCL_DOSQUEUES

#define ENABLE_EXCEPTION 2
#define ENABLE_HARDERROR 1
#define DISABLE_ERRORPOPUIS 0
#define ENABLE_ERRORPOPUIS ENABLE_EXCEPTION | ENABLE_HARDERROR

USHORT rc;

rc = DosError(DISABLE_ERRORPOPUIS); /* Action flag */
rc = DosError(ENABLE_ERRORPOPUIS); /* Action flag */
```

This call allows a program to request that another program execute as a child process.

**DosExecPgm** (*ObjNameBuf*, *ObjNameBufL*, *ExecFlags*, *ArgPointer*, *EnvPointer*, *ReturnCodes*, *PgmPointer*)

## Parameters

**ObjNameBuf** (*PCHAR*) — output

Address of the name of the object that contributed to the failure of DosExecPgm is returned.

**ObjNameBufL** (*SHORT*) — input

Length, in bytes, of the buffer described by ObjNameBuf.

**ExecFlags** (*USHORT*) — input

Indicates how the program executes in relation to the requestor and whether execution is under conditions for debugging.

Value	Definition
0	Execution is synchronous to the parent process. The termination code and result code are stored in the two-word structure.
1	Execution is asynchronous to the parent process. When the child process terminates, its result code is discarded. The process ID is stored in the first word of the two-word structure ReturnCodes.
2	Execution is asynchronous to the parent process. When the child process terminates, its result code is saved for examination by a DosCwait request. The process ID is stored in the first word of the two-word structure ReturnCodes.
3	Execution is the same as if ExecFlags=2 is specified, plus debugging conditions are present for the child process.
4	Execution is asynchronous to and detached from the parent process session. When the detached process is started, it is not affected by the ending of the parent process.
5	The program is loaded into storage and made ready to execute, but is not placed into execution until the session manager dispatches the threads belonging to the process.
6	Execution is the same as if ExecFlag=2 is specified, with the addition of debugging conditions being present for the child process and any of its descendants.

Some memory is consumed for uncollected result codes. Issue DosCwait to release this memory. If result codes are not collected, then ExecFlags=0 or 1 should be used.

**ArgPointer** (*PSZ*) — input

Address of the ASCIIZ Argument strings passed to the program. These strings represent command parameters, which are copied to the environment segment of the new process. The convention used by CMD.EXE is that the first of these strings is the program name (as entered from the command prompt or found in a batch file), and the second string consists of parameters to the program name. The second ASCIIZ string is followed by an additional byte of zeros. A value of 0 for the address of ArgPointer means that no arguments are to be passed.

**EnvPointer**(*PSZ*) — input

Address of the ASCIIZ environment strings passed to the program. These strings represent environment variables and their current values. An environment string has the following form:

variable=value

The last ASCIIZ environment string must be followed by an additional byte of zeros.

A value of 0 for the address of EnvPointer results in the new process inheriting the environment of its parent process.

When the new process is given control, it receives:

- A pointer to its environment segment
- The fully qualified path name of the executable file
- A copy of the argument strings.

A coded example of this follows:

```
eo:  ASCIIZ string 1 ; environment string 1
      ASCIIZ string 2 ; environment string 2
      :
      ASCIIZ string n ; environment string n
      Byte of 0
      :
po:  ASCIIZ          ; string of filename
      ; of program to run.
      :
ao:  ASCIIZ          ; argument string 1
      ASCIIZ          ; argument string 2
      Byte of 0
```

The beginning of the environment segment is "eo" and "ao" is the offset of the first argument string in that segment. Register BX contains "ao" on entry to the target program. The address to the environment segment can also be obtained by issuing DosGetInfoSeg.

## **ReturnCodes (PRESULTCODES) — output**

Address of the structure containing the process ID or termination code and the result code indicating the reason for the child's termination. This structure is also used by a DosCwait request, which waits for an asynchronous child process to end.

### **termcodepid (USHORT)**

For an asynchronous request, the process identifier of the child process. For a synchronous request, the termination code furnished by the system describes why the child terminated.

Value	Definition
0	EXIT (normal)
1	Hard error abort
2	Trap operation
3	Unintercepted DosKillProcess

### **resultcode (USHORT)**

Result code specified by the terminating synchronous process on its last DosExit call.

## **PgmPointer (PSZ) — input**

Address of the name of the file that contains the program to be executed. When the environment is passed to the target program, this name is copied into "po" in the environment description shown above.

If the ASCIIZ string appears to be a fully qualified path (contains a ":" in the second position and/or contains a "\"), only the indicated drive:directory is searched. If the string is not a fully qualified path, the current directory is searched. If the file name is not found in the current directory, each drive:directory specification in the PATH defined in the current process' environment is searched for this file. Note that any extension (.XXX) is acceptable for the executable file being loaded.

## **rc (USHORT) — return**

Return code descriptions are:

0	NO_ERROR
1	ERROR_INVALID_FUNCTION
2	ERROR_FILE_NOT_FOUND
3	ERROR_PATH_NOT_FOUND
4	ERROR_TOO_MANY_OPEN_FILES
5	ERROR_ACCESS_DENIED
8	ERROR_NOT_ENOUGH_MEMORY
10	ERROR_BAD_ENVIRONMENT
11	ERROR_BAD_FORMAT
13	ERROR_INVALID_DATA

26	ERROR_NOT_DOS_DISK
32	ERROR_SHARING_VIOLATION
33	ERROR_LOCK_VIOLATION
36	ERROR_SHARING_BUFFER_EXCEEDED
89	ERROR_NO_PROC_SLOTS
95	ERROR_INTERRUPT
108	ERROR_DRIVE_LOCKED
127	ERROR_PROC_NOT_FOUND
180	ERROR_INVALID_SEGMENT_NUMBER
182	ERROR_INVALID_ORDINAL
188	ERROR_INVALID_STARTING_CODESEG
189	ERROR_INVALID_STACKSEG
190	ERROR_INVALID_MODULETYPE
191	ERROR_INVALID_EXE_SIGNATURE
192	ERROR_EXE_MARKED_INVALID
194	ERROR_ITERATED_DATA_EXCEEDS_64k
195	ERROR_INVALID_MINALLOCSIZE
196	ERROR_DYNLINK_FROM_INVALID_RING
198	ERROR_INVALID_SEGDPL
199	ERROR_AUTODATASEG_EXCEEDS_64k
201	ERROR_RELOC_CHAIN_XEEDS_SEGLIM

**Remarks**

The target program is located and loaded into storage if necessary. A process is created and executed for the target program. The new process is created with an address space separate from its parent; that is, a new Local Descriptor Table (LDT) is built for the process.

The execution of a child process can be synchronous or asynchronous to the execution of its parent process. If synchronous execution is indicated, the requesting thread waits pending completion of the child process. Other threads in the requesting process may continue to run.

If asynchronous execution is indicated, DosExecPgm returns with the process ID of the started child process. Specifying ExecFlags=2 allows the parent process to issue a DosCwait request after the DosExecPgm request, so it can examine the result code returned when the child process terminates. If ExecFlags=1 is specified, the result code of the asynchronous child process is not returned to the parent process.

A child process inherits file handles obtained by its parent with DosOpen calls that indicated inheritance. The child process also inherits handles to pipes created by the parent process with DosMakePipe.

Because a child process has the ability to inherit handles and a parent process controls the meanings of handles for standard I/O, the parent can duplicate inherited handles as handles for standard I/O. This permits the parent process and the child process to coordinate I/O to a pipe or a file.

For example, a parent process can create two pipes with DosMakePipe requests. It can issue DosDupHandle to redefine the read handle of one pipe as standard input (0000H), and the write handle of the other pipe as standard output (0002H). The child process uses the standard I/O handles, and the parent process uses the remaining read and write pipe handles. Thus, the child process reads what the parent writes to one pipe, and the parent process reads what the child writes to the other pipe.

When an inherited file handle is duplicated, the position of the file pointer is always the same for both handles, regardless of which handle repositions the file pointer.

An asynchronous process started with ExecFlags=3 or ExecFlags=6 is provided a trace flag facility. This facility and the Ptrace buffer provided by DosPtrace enable a debugger to perform breakpoint debugging. DosStartSession provides additional debugging capabilities that allow a debugger to trace all processes associated with an application running in a child session, regardless of whether the process is started with a DosExecPgm or a DosStartSession request.

A detached process is treated as an orphan of the parent process and executes in the background. Thus, it cannot make any VIO, KBD, or MOU calls, except within a video pop-up requested by VioPopUp. To test whether a program is running detached, use the following method. Issue a video call, (for example, VioGetAnsi). If the call is not issued within a video pop-up and the process is detached, the video call returns error code ERROR\_VIO\_DETACHED.

**Note:** If the target program's entry point is in a segment that has IOPL indicated, this causes a general protection fault and the process is terminated. If any dynamic link module being used by the new process has an initialization routine specified in a segment that has IOPL indicated, general protection fault occurs and the process is terminated.

### Family API Considerations

Some options operate differently in DOS mode than in OS/2 mode. Therefore, the following restrictions apply to DosExecPgm when coding in DOS mode:

- ExecFlags must be set to zero. This value is not related to the PID of the program being executed. If ExecFlags  $\neq$  0, DosExecPgm returns the error code ERROR\_INVALID\_DATA.
- The ObjNameBuf field is used to provide additional information in the OS/2 mode environment as to why the DosExecPgm failed. The information is not relevant or available in DOS 2.X or DOS 3.X. Therefore, the buffer is filled in with blanks.
- The ReturnCodes two-word structure is very similar to the OS/2 mode environment. The first word is a termination code with the following meanings:

Value	Definition
0	Exit (normal exit and termination by call INT 21H AH=31H)
1	Hard error abort
2	Not returned
3	Termination by Ctrl-Break.

The second word contains the ResultCode specified by the terminating process on its DosExit call (or INT 21H AH=4CH call).

### C Language

```
typedef struct _RESULTCODES { /* resc */

    USHORT codeTerminate;      /* Termination Code -or- Process ID */
    USHORT codeResult;         /* Exit Code */

} RESULTCODES;

#define INCL_DOSPROCESS

USHORT rc = DosExecPgm(ObjNameBuf, ObjNameBufL, ExecFlags, ArgPointer,
                      EnvPointer, ReturnCodes, PgmPointer);

PCHAR    ObjNameBuf; /* Address of object name buffer (returned) */
SHORT    ObjNameBufL; /* Length of object name buffer */
USHORT    ExecFlags; /* Execute asynchronously/trace */
PSZ       ArgPointer; /* Address of argument string */
PSZ       EnvPointer; /* Address of environment string */
RESULTCODES ReturnCodes; /* Address of termination codes (returned) */
PSZ       PgmPointer; /* Address of program file name */

USHORT    rc; /* return code */
```

## Assembler Language

```

RESULTCODES struc

    resc_codeTerminate dw ? ;Termination Code -or- Process ID
    resc_codeResult    dw ? ;Exit Code

RESULTCODES ends

EXTRN DosExecPgm:FAR
INCL_DOSPROCESS EQU 1

PUSH@ OTHER ObjNameBuf ;Object name buffer (returned)
PUSH WORD ObjNameBufL ;Length of object name buffer
PUSH WORD ExecFlags ;Execute asynchronously/trace
PUSH@ ASCIIZ ArgPointer ;Address of argument string
PUSH@ ASCIIZ EnvPointer ;Address of environment string
PUSH@ DWORD ReturnCodes ;Termination codes (returned)
PUSH@ ASCIIZ PgmPointer ;Program file path name string
CALL DosExecPgm

Returns WORD

```

## Example

This example starts up the program simple.exe and then waits for it to finish. Then the termination and return codes are printed.

```

#define INCL_DOSPROCESS

#define START_PROGRAM "simple.exe"

CHAR LoadError[100];
PSZ Args;
PSZ Envs;
RESULTCODES ReturnCodes;
USHORT rc;

if(!DosExecPgm(LoadError, /* Object name buffer */
               sizeof(LoadError), /* Length of object name buffer */
               EXEC_SYNC, /* Asynchronous/Trace flags */
               Args, /* Argument string */
               Envs, /* Environment string */
               &ReturnCodes, /* Termination codes */
               START_PROGRAM)) /* Program file name */
    printf("Termination Code %d Return Code %d \n",
           ReturnCodes.codeTerminate,
           ReturnCodes.codeResult);

-----simple.exe-----

#define INCL_DOSPROCESS

#define RETURN_CODE 0

main( )
{
    printf("Hello!\n");
    DosExit(EXIT_PROCESS, /* End thread/process */
           RETURN_CODE); /* Result code */
}

```

The following example demonstrates how to create a process, obtain process ID information, and kill a process. Process1 invokes process2 to run asynchronously. It obtains and prints some PID information, and then kills process2.

# DosExecPgm – Execute Program

FAP1

```
/* ---- process1.c ---- */

#define INCL_DOSPROCESS

#include <os2.h>

#define START_PROGRAM "process2.exe" /* Program pointer */

main()
{
    CHAR        ObjFail [50];        /* Object name buffer */
    RESULTCODES ReturnCodes;          /*
    PIDINFO      PidInfo;             /*
    PID          ParentID;            /*
    USHORT       rc;

    printf("Process1 now running. \n");

    /** Start a child process. **/
    if(!DosExecPgm(ObjFail,           /* Object name buffer */
                  sizeof(ObjFail),    /* Length of obj. name buffer */
                  EXEC_ASYNC,         /* Execution flag - asynchronous */
                  NULL,               /* No args. to pass to process2 */
                  NULL,               /* Process2 inherits process1's environment */
                  &ReturnCodes,       /* Ptr. to resultcodes struct. */
                  START_PROGRAM)))    /* Name of program file */
        printf("Process2 started. \n");

    /** Obtain Process ID information and print it **/
    if(!rc=DosGetPID(&PidInfo))       /* Process ID's (returned) */
        printf("DosGetPID: current process ID is %d; thread ID is %d; parent process ID is %d.\n",
               PidInfo.pid, PidInfo.tid, PidInfo.pidParent);
    if(!rc=DosGetPPID(
        ReturnCodes.codeTerminate, /* Process whose parent is wanted */
        &ParentID))               /* Address to put parent's PID */
        printf("Child process ID is %d; Parent process ID is %d.\n",
               ReturnCodes.codeTerminate, ParentID);

    /** Terminate process2 **/
    if(!rc=DosKillProcess(DKP_PROCESSTREE, /* Action code - kill process and descendants */
                          ReturnCodes.codeTerminate)) /* PID of root of process tree */
        printf("Process2 terminated by process1.\n");
}

/* ---- process2.c ---- */

#define INCL_DOSPROCESS

#include <os2.h>

#define SLEEPTIME 500L
#define RETURN_CODE 0

main()
{
    printf("Process2 now running.\n");

    /** Sleep to allow process1 to kill it */
    DosSleep(SLEEPTIME);             /* Sleep interval */
    DosExit(EXIT_PROCESS,            /* Action Code */
            RETURN_CODE);            /* Result Code */
}
```

---

This call is issued when a thread completes executing. The current thread or process ends.

<b>DosExit (ActionCode, ResultCode)</b>
---

## Parameters

**ActionCode (USHORT)** – input

Terminates the process and all its threads.

Value	Definition
0	The current thread ends.
1	All threads in the process end.

**ResultCode (USHORT)** – input

Program's completion code. It is passed to any thread that issues DosCwait for this process.

## Remarks

DosExit allows a thread to terminate itself or be terminated by another thread in its process. If ActionCode=0 and the specified thread is the last thread executing in the process, or if ActionCode=1, the process terminates.

The system can start threads on behalf of an application. Thus, if the intent of a DosExit call is to terminate the process, ActionCode=1 should be specified to terminate all the threads belonging to the process.

Do not terminate thread 1 without terminating the process. Thread 1 is the initial thread of execution, not a thread started by a DosCreateThread request. When thread 1 ends, any monitors or signal processing routines set for this process also end. To avoid unpredictable results, DosExit should be specified with ActionCode=1 to ensure the process ends.

When a process is terminating, all but one thread is terminated and that thread executes routines whose addresses have been specified with DosExitList. After resources have been cleaned up by the exit list routines, this thread and all other resources owned by the process are released.

## Family API Considerations

Some options operate differently in the DOS mode than in the OS/2 mode. Therefore, the following restrictions apply to DosExit when coding for the DOS mode:

There is no thread support in DOS 3.3; therefore DosExit exits the currently executing program.

If ActionCode = 0 this option is ignored. It is equivalent to an ActionCode = 1.

## C Language

```
#define INCL_DOSPROCESS
```

```
VOID DosExit(ActionCode, ResultCode);
```

```
USHORT ActionCode; /* Indicates end thread or process */
USHORT ResultCode; /* Result Code to save for DosCwait */
```



# DosExit — Exit Program

FAPI

## Assembler Language

```
EXTRN DosExit:FAR
INCL_DOSPROCESS EQU 1

PUSH WORD   ActionCode   ;Indicates end thread or process
PUSH WORD   ResultCode   ;Result Code to save for DosCwait
CALL DosExit
```

## Example

In this example, the main routine starts up another program, simple.exe, and then expects a return code of 3 to be returned. Simple.exe sets the return code with DosExit.

```
#define INCL_DOSPROCESS

#define START_PROGRAM "simple.exe"
#define RETURN_OK 3

CHAR      LoadError[100];
PSZ       Args;
PSZ       Envs;
RESULTCODES ReturnCodes;
USHORT    rc;

    if(!DosExecPgm(LoadError,                /* Object name buffer */
                   sizeof(LoadError),        /* Length of object name buffer */
                   EXEC_SYNC,                /* Asynchronous/Trace flags */
                   Args,                     /* Argument string */
                   Envs,                     /* Environment string */
                   &ReturnCodes,            /* Termination codes */
                   START_PROGRAM))           /* Program file name */
    if (ReturnCodes.codeResult == RETURN_OK) /* Check result code */
        printf("things are ok..");
    else
        printf("something is wrong...");

-----simple.exe-----

#define INCL_DOSPROCESS

#define RETURN_CODE 3

main( )
{
    printf("Hello!\n");
    DosExit(EXIT_THREAD, /* End thread/process */
            RETURN_CODE); /* Result code */
}
```

The following example shows how to suspend and resume execution of a thread within a process. The main thread creates Thread2 and allows it to begin executing. Thread2 iterates through a loop that prints a line and then sleeps, relinquishing its time slice to the main thread. After one iteration by Thread2, the main thread suspends Thread2 and then resumes it. Subsequently, Thread2 completes the remaining three iterations.

```
#define INCL_DOSPROCESS

#include <os2.h>

#define SEGSIZE      4096  /* Number of bytes requested in segment */
#define ALLOCFLAGS    0    /* Segment allocation flags - no sharing */
#define SLEEPSHORT    5L   /* Sleep interval - 5 milliseconds */
#define SLEEPLONG     75L  /* Sleep interval - 75 milliseconds */
#define RETURN_CODE    0    /* Return code for DosExit() */

VOID APIENTRY Thread2()
{
    USHORT    i;

    /* Loop with four iterations */
    for(i=1; i<5; i++)
    {
        printf("In Thread2, i is now %d\n", i);

        /* Sleep to relinquish time slice to main thread */
        DosSleep(SLEEPSHORT);      /* Sleep interval */
    }
    DosExit(EXIT_THREAD,          /* Action code - end a thread */
            RETURN_CODE);         /* Return code */
}

main()
{
    TID        ThreadID;          /* Thread identification */
    SEL        ThreadStackSel;    /* Segment selector for thread stack */
    PBYTE      StackEnd;          /* Ptr. to end of thread stack */
    USHORT     rc;

    /** Allocate segment for thread stack; make pointer to end of stack. **/
    /** We must allocate a segment in order to preserve segment protection **/
    /** for the thread. **/

    rc = DosAllocSeg(SEGSIZE,      /* Number of bytes requested */
                    &ThreadStackSel, /* Segment selector (returned) */
                    ALLOCFLAGS);    /* Allocation flags - no sharing */
    StackEnd = MAKEP(ThreadStackSel, SEGSIZE-1);

    /** Start Thread2 **/
    if(!rc=DosCreateThread((PFNTHREAD) Thread2, /* Thread address */
                          &ThreadID,           /* Thread ID (returned) */
                          StackEnd))) /* End of thread stack */
        printf("Thread2 created.\n");

    /* Sleep to relinquish time slice to Thread2 */
    if(!(DosSleep(SLEEPSHORT))) /* Sleep interval */
        printf("Slept a little to let Thread2 execute.\n");

    /***** Suspend Thread2, do some work, then resume Thread2 *****/
    if(!rc=DosSuspendThread(ThreadID)) /* Thread ID */
        printf("Thread2 SUSPENDED.\n");
    printf("Perform work that will not be interrupted by Thread2.\n");
    if(!rc=DosResumeThread(ThreadID)) /* Thread ID */
        printf("Thread2 RESUMED.\n");
    printf("Now we may be interrupted by Thread2.\n");

    /* Sleep to allow Thread2 to complete */
    DosSleep(SLEEPLONG); /* Sleep interval */
}
```

# DosExitCritSec —

## Exit Critical Section of Execution

---

This call restores normal thread dispatching for the current process.

**DosExitCritSec ( )**

### Parameters

None

**rc** (*USHORT*) — return

Return code descriptions are:

0	NO_ERROR
485	ERROR_CRITSEC_UNDERFLOW

### Remarks

A count is maintained of the number of times a `DosEnterCritSec` request is made without a corresponding `DosExitCritSec`. The count is incremented by `DosEnterCritSec` and decremented by `DosExitCritSec`. Normal thread dispatching is not restored until the count is 0.

The outstanding `DosEnterCritSec` count is maintained in a word. If overflow occurs, the count is set to the maximum value, no operation is performed, and the request returns with an error.

### C Language

```
#define INCL_DOSPROCESS

USHORT rc = DosExitCritSec(VOID);

USHORT rc; /* return code */
```

### Assembler Language

```
EXTRN DosExitCritSec:FAR
INCL_DOSPROCESS EQU 1

CALL DosExitCritSec

Returns WORD
```

### Example

This example enters a section that will not be pre-empted, performs a simple task, and then exits quickly.

```
#define INCL_DOSPROCESS

DosEnterCritSec(); /* Enter critical code section */
flag = TRUE; /* Perform some work */
DosExitCritSec(); /* Exit critical code section */
```

# DosExitList – Maintains Routine List for Process Termination

This call maintains a list of routines that execute when the current process ends.

**DosExitList (FcnCode\_Order, RtnAddress)**

## Parameters

**FcnCode\_Order (USHORT)** – input

Two-byte fields. The low-order byte indicates the operation being performed by DosExitList, which can be used to update the list of routines, or to transfer to the next address on the termination list at the completion of a routine. The values of the byte and their meanings are:

Value	Definition
1	Add address to termination list.
2	Remove address from termination list.
3	Transfer to next address on termination list.

The high-order byte indicates the invocation order. This value is valid only when the low-order byte is 1 (add an address). For the other low-order byte values, the high-order byte must be set to zero.

The invocation order determines the order in which routines are invoked. Routines given a value of 0 are invoked first and routines with a value of 255 are invoked last. If more than one routine is added with the same invocation order value, the last routine to be added is invoked first. The following values are used by OS/2 components:

Value	Definition
80H – 88H	OS/2 Extended Edition Database Manager
90H – 98H	OS/2 Extended Edition Communications Manager
A0H – A8H	OS/2 Presentation Manager
B0H	OS/2 KBD component
C0H	OS/2 VIO component
D0H	OS/2 IPC Queues component

**RtnAddress (PFNEXITLIST)** – input

The address of a routine to be executed. This address cannot be in an IOPL segment.

**rc (USHORT)** – return

Return code descriptions are:

0	NO_ERROR
8	ERROR_NOT_ENOUGH_MEMORY
13	ERROR_INVALID_DATA

## Remarks

DosExitList maintains a list of addresses to routines that receive control when a process is terminated. These addresses must be in the address space of the terminating process. DosExitList routines perform clean-up operations on resources. For example, DosExitList can be used in a dynamic link library module to free resources and semaphores after a client program has ended.

During DosExitList processing, the process is in a state of partial termination. All threads of the process are terminated, except for the one executing the routines. Ownership of any exclusive semaphores created by the process with DosCreateSem was transferred to the DosExitList thread, so the thread can access protected resources. Termination routines should be short and fail-safe, so there is minimum delay in completing process termination.

Before transferring control to the termination routines, OS/2 resets the stack to its initial value. Transfer is by way of a JMP instruction. The first parameter on the stack (located at SS:SP+4) contains an indicator of why the process ended. The meanings of values returned are the same as those for termination codes returned by DosCwait or DosExecPgm requests. These values are:

0	EXIT (normal exit)
---	--------------------

# DosExitList —

## Maintains Routine List for Process Termination

- 1            Hard error abort
- 2            Trap operation
- 3            Unintercepted DosKillProcess.

Each routine on the list receives control in numeric order by function high-order byte. For example, low (0) is first with high (OFFH) being last. In case of duplicate entries for the same value, the routines are executed in LIFO order.

When a routine has completed its processing, it issues DosExitList with function = 3. Control is then transferred to the next address in the invocation order. If a routine on the list does not issue DosExitList at the completion of its processing, the process hangs, and OS/2 prevents termination. When all addresses are serviced, the process completes exiting.

Most OS/2 system calls are valid in a DosExitList routine; however, certain functions such as DosCreateThread and DosExecPgm are not. Functions requested in a routine must not be higher in the function code order hierarchy than the invocation order specified for the routine.

### C Language

```
#define INCL_DOSPROCESS

USHORT rc = DosExitList(FcnCode_Order, RtnAddress);

USHORT      FcnCode_Order; /* Function request code/Order */
PFNEXITLIST RtnAddress;    /* Address of routine */

USHORT      rc;            /* return code */
```

### Assembler Language

```
EXTRN DosExitList:FAR
INCL_DOSPROCESS EQU 1

PUSH WORD FcnCode_Order ;Function request code/Order
PUSH DWORD RtnAddress ;Address of routine
CALL DosExitList
```

Returns WORD

### Example

In this example, TestRoutine is added to the exitlist sequence. Routines in the exitlist sequence must use DosExitList instead of DosExit to end.

```
#define INCL_DOSPROCESS
#define INCL_VIO
#define ROUTINE_ORDER 0xEE00
#define VIO_HANDLE 0

USHORT rc;

VOID APIENTRY TestRoutine2()
{
    USHORT r;

    VioWrtTTY("This runs last...\n", /* String to be written */
              18, /* Length of string */
              VIO_HANDLE); /* Video handle */
    r = DosExitList(EXLST_EXIT, /* Function request code/order */
                   (PFNEXITLIST) TestRoutine2); /* Address of routine */
}

main()
{
    rc = DosExitList(EXLST_ADD | ROUTINE_ORDER, /* Function request code/order */
                   (PFNEXITLIST) TestRoutine2); /* Address of routine */
}
```

This call performs multiple locking, unlocking, seeking, and I/O operations on an opened file.

**DosFileIO (FileHandle, CommandList, CommandListLen, ErrorOffset)**

## Parameters

**FileHandle (HFILE)** – input  
The handle of the file.

**CommandList (PBYTE)** – input  
Address of the list of entries, indicating the operations to be performed. CmdLock, CmdUnlock, CmdSeek, and CmdIO operations are atomic. Thus, the completion of one operation is not dependent upon the completion of another operation following it in the list. Operations are performed until all are complete or until one fails.

### CmdLock

Lock command structure. To perform lock operations on one or more regions in a file, a structure is created that has the following format:

**Cmd (USHORT)** – input  
A value of 0 is required for a lock operation.

**LockCnt (USHORT)** – input  
Number of regions in the file that are to be locked.

**TimeOut (LONG)** – input  
The count in milliseconds until the requesting process is to resume execution if the requested locks are not available. In addition to specifying a milliseconds value, the following values and their meanings may also be specified:

Value	Definition
FFFFFFFFH	Wait indefinitely until the requested locks become available.
00000000H	Return immediately to the requesting process.

Because the lock operation is atomic, if a lock within a CmdLock causes a time out, none of the other locks within the scope of CmdLock are in force.

### Lock

Lock record structure. The CmdLock structure is followed by a series of records in the following format:

**Share (USHORT)** – input  
Defines the type of access other processes may have to the file range being locked. Values and their meanings are:

Value	Definition
0	Other processes have "No-Access" to the locked range. The current process has both read and write access to the locked range. A region with Share = 0 must not overlap any other locked region.
1	Other processes and the current process have "Read-Only" access to the locked range. A range locked with Share = 1 may overlap other ranges locked with Share = 1, but must not overlap other ranges locked with Share = 0.

**Start (ULONG)** – input  
Offset into file where region to be locked starts.

**Length (ULONG)** – input  
Length of region to be locked.

# DosFileIO –

## Perform File I/O

### CmdUnlock

Unlock command structure. To perform unlock operations on one or more regions in a file, a structure is created that has the following format:

**Cmd** (*USHORT*) – input

A value of 1 is required for an unlock operation.

**UnlockCnt** (*USHORT*) – input

Number of regions in the file that are to be unlocked.

### UnLock

Unlock record structure. The CmdUnlock structure is followed by a series of records in the following format:

**Start** (*ULONG*) – input

Offset into file where region to be unlocked starts.

**Length** (*ULONG*) – input

Length of region to be unlocked.

### CmdSeek

Seek command structure. To perform seek operations on one or more locked regions in a file, a structure is created that has the following format:

**Cmd** (*USHORT*) – input

A value of 2 is required for a seek operation.

**Method** (*USHORT*) – input

Location in file whose offset is used to calculate the new position of the file pointer by adding to it the offset specified by Position.

Value	Definition
0	Beginning of the file
1	Current location of the file pointer
2	End of the file.

**Position** (*LONG*) – input

The distance to move, in bytes.

**Actual** *ULONG* – output

New position of the file pointer.

### CmdIO

I/O command structure. To perform I/O operations on one or more locked regions in a file, a structure is created that has the following format:

**Cmd** (*USHORT*) – input

Either of the following values are specified:

Value	Definition
3	Read operation
4	Write operation.

**Buffer** (*PBYTE*) – input/output

Address of the input or output buffer.

**BufferLen** (*USHORT*) – input

Number of bytes requested to be read or written.

**Actual** (*USHORT*) – output

Number of bytes actually transferred.

**CommandListLen** (*USHORT*) – input

The length in bytes of CommandList.

**ErrorOffset** (*PLONG*) – output

Address of the byte offset of the record in which the error occurred.

## DosFileIO – Perform File I/O

**rc** (*USHORT*) – return

Return code descriptions are:

0	NO_ERROR
5	ERROR_ACCESS_DENIED
6	ERROR_INVALID_HANDLE
33	ERROR_LOCK_VIOLATION
36	ERROR_SHARING_BUFFER_EXCEEDED
87	ERROR_INVALID_PARAMETER
95	ERROR_INTERRUPT
130	ERROR_DIRECT_ACCESS_HANDLE
131	ERROR_NEGATIVE_SEEK
132	ERROR_SEEK_ON_DEVICE

### Remarks

DosFileIO combines the following operations into a single request:

- Unlocking and locking multiple file ranges
- Changing the file position pointer
- Performing file I/O.

The ability to combine operations on files provides improved performance and makes DosFileIO particularly suited to a networking environment.

Unlike DosFileLocks, which unconditionally prevents access to only one range in a file at a time, DosFileIO permits multiple regions to be locked. DosFileIO also offers the option of read-only access to locked regions by the current process and any sharing processes.

If another process attempts to read or write in a “No-access” region, or if any process attempts to write in a “Read-only” region, ERROR\_ACCESS\_DENIED is returned.

A range to be locked must first be cleared of any locked subranges or overlapping ranges. If a time out occurs before locking can be completed, DosFileIO returns with an unsuccessful return code. The caller may return after the time-out period has expired without receiving an ERROR\_SEM\_TIMEOUT. Therefore, semaphore time-out values should not be used for exact timing and sequencing.

### C Language

```
#define INCL_DOSFILEMGR
```

```
USHORT rc = DosFileIO(FileHandle, CommandList, CommandListLen, ErrorOffset);
```

```
HFILE      FileHandle;    /* File handle */
PBYTE      CommandList;   /* Ptr to command buffer */
USHORT     CommandListLen; /* Length of command buffer */
PLONG      ErrorOffset;    /* Ptr to command error offset */

USHORT     rc;             /* return code */
```

### Assembler Language

```
EXTRN DosFileIO:FAR
INCL_DOSFILEMGR EQU 1

PUSH WORD FileHandle ;File handle
PUSH@ OTHER CommandList ;Command buffer
PUSH WORD CommandListLen ;Length of command buffer
PUSH@ WORD ErrorOffset ;Command error offset (returned)
CALL DosFileIO
```

Returns WORD



---

This call locks and unlocks a range in an opened file.

<b>DosFileLocks</b> ( <i>FileHandle</i> , <i>UnLockRange</i> , <i>LockRange</i> )
---

## Parameters

**FileHandle** (*HFILE*) — input  
File handle.

**UnLockRange** (*PLONG*) — input  
Address of the structure containing the offset and length of a range to be unlocked. A doubleword of zero indicates that unlocking is not required.

**FileOffset** (*ULONG*)  
The offset to the beginning of the range to be unlocked.

**RangeLength** (*ULONG*)  
The length of the range to be unlocked.

**LockRange** (*PLONG*) — input  
Address of the structure containing the offset and length of a range to be locked. A doubleword of zero indicates that locking is not required.

**FileOffset** (*ULONG*)  
The offset to the beginning of the range to be locked.

**RangeLength** (*ULONG*)  
The length of the range to be locked.

**rc** (*USHORT*) — return  
Return code descriptions are:

0	NO_ERROR
6	ERROR_INVALID_HANDLE
33	ERROR_LOCK_VIOLATION
36	ERROR_SHARING_BUFFER_EXCEEDED

## Remarks

DosFileLocks provides a mechanism that allows a process to lock a region in a file for read/write access. The time a region is locked should be short.

Instead of denying another process read/write access to the entire file by means of access and sharing modes specified with DosOpen or DosOpen2 requests, a process attempts to lock only the the range needed for read/write access and examines the error code returned.

A range to be locked must first be cleared of any locked subranges or overlapping ranges. The locked region can be located anywhere in the file, and locking beyond end-of-file is not considered an error.

Once a lock is successful, read/write access by another process to the specified range is denied until the range is unlocked. If both unlocking and locking are specified by a DosFileLocks request, the unlocking operation is performed first. After unlocking is completed, locking is done.

Duplicating the handle duplicates access to any locked regions; however, access to locked regions is not duplicated across the DosExecPgm call.

If a file is closed (either by a DosClose request or by a process terminating) and locks are still in effect, the locks are released in no defined order.

## Family API Considerations

Some options operate differently in the DOS mode than in OS/2 mode. Therefore, the following restrictions apply to DosFileLocks when coding for the DOS mode:

- If Block = 1 is specified, an "invalid range lock list" or "invalid unlock list" error is returned.
- NewLockIDList is not supported.

## C Language

```
#define INCL_DOSFILEMGR

USHORT rc = DosFileLocks(FileHandle, UnLockRange, LockRange);

HFILE      FileHandle;    /* File handle */
PLONG      UnLockRange;   /* UnLock range */
PLONG      LockRange;     /* Lock range */

USHORT      rc;           /* return code */
```

## Assembler Language

```
EXTRN DosFileLocks:FAR
INCL_DOSFILEMGR EQU 1

PUSH WORD FileHandle ;File handle
PUSH@ OTHER UnLockRange ;UnLock range
PUSH@ OTHER LockRange ;Lock range
CALL DosFileLocks

Returns WORD
```

# DosFileLocks — Manages File Locking

FAPI

## Example

This example opens a file, writes some data to it, locks a block of the data, and then unlocks it.

```
#define INCL_DOSFILEMGR

#define OPEN_FILE 0x01
#define CREATE_FILE 0x10
#define FILE_ARCHIVE 0x20
#define FILE_EXISTS OPEN_FILE
#define FILE_NOEXISTS CREATE_FILE
#define DASD_FLAG 0
#define INHERIT 0x80
#define WRITE_THRU 0
#define FAIL_FLAG 0
#define SHARE_FLAG 0x10
#define ACCESS_FLAG 0x02

#define FILE_NAME "test.dat"
#define FILE_SIZE 800L
#define FILE_ATTRIBUTE FILE_ARCHIVE
#define RESERVED 0L
#define NULL_RANGE 0L

HFILE FileHandle;
USHORT Wrote;
USHORT Action;
PSZ FileData[100];
USHORT rc;

struct LockStrc
{
    long Offset;
    long Range;
} Area;

int i;

Action = 2;
strcpy(FileData, "Data...");
Area.Offset = 4;
Area.Range = 100;

if(!DosOpen(FILE_NAME, /* File path name */
             &FileHandle, /* File handle */
             &Action, /* Action taken */
             FILE_SIZE, /* File primary allocation */
             FILE_ATTRIBUTE, /* File attribute */
             FILE_EXISTS | FILE_NOEXISTS, /* Open function type */
             DASD_FLAG | INHERIT, /* Open mode of the file */
             WRITE_THRU | FAIL_FLAG |
             SHARE_FLAG | ACCESS_FLAG,
             RESERVED)) /* Reserved (must be zero) */
{
    for(i=0; i<200; ++i)
        DosWrite(FileHandle, /* File handle */
                 FileData, /* User buffer */
                 sizeof(FileData), /* Buffer length */
                 &Wrote); /* Bytes written */
    rc = DosFileLocks(FileHandle, /* File handle */
                     NULL_RANGE, /* Unlock range */
                     (PLONG) &Area); /* Lock range */
    rc = DosFileLocks(FileHandle, /* File handle */
                     (PLONG) &Area, /* Unlock range */
                     NULL_RANGE); /* Lock range */
}
```

This call closes the association between a directory handle and a DosFindFirst or DosFindNext directory search function.

**DosFindClose (DirHandle)**

## Parameters

**DirHandle** (*HDIR*) — input

Handle previously associated with a DosFindFirst by the system, or used with a DosFindNext directory search function.

**rc** (*USHORT*) — return

Return code descriptions are:

0	NO_ERROR
6	ERROR_INVALID_HANDLE

## Remarks

When DosFindClose is issued, a subsequent DosFindNext for the closed DirHandle fails unless an intervening DosFindFirst has been issued specifying DirHandle.

## C Language

```
#define INCL_DOSFILEMGR

USHORT rc = DosFindClose(DirHandle);

HDIR      DirHandle;    /* Directory search handle */

USHORT     rc;           /* return code */
```

## Assembler Language

```
EXTRN DosFindClose:FAR
INCL_DOSFILEMGR EQU 1

PUSH WORD DirHandle ;Directory search handle
CALL DosFindClose
```

Returns WORD

## Example

This example searches for a file, then closes the search.

```
#define INCL_DOSFILEMGR
#define SEARCH_PATTERN "*.*"
#define FILE_ATTRIBUTE 0
#define RESERVED 0L

HDIR FindHandle;

FindHandle = 0x0001;
FindCount = 1;

rc = DosFindFirst(SEARCH_PATTERN,      /* File pattern */
                  &FindHandle,        /* Directory search handle */
                  FILE_ATTRIBUTE,      /* Search attribute */
                  &FindBuffer,        /* Result buffer */
                  sizeof(FindBuffer), /* Result buffer length */
                  &FindCount,         /* # of entries to find */
                  RESERVED);           /* Reserved (must be zero) */
rc = DosFindClose(FindHandle);        /* Directory search handle */
```

# DosFindFirst —

## Find First Matching File Object

FAPI

This call finds the first file object or group of file objects whose name(s) match the specification.

**DosFindFirst** (**FileName**, **DirHandle**, **Attribute**, **ResultBuf**, **ResultBufLen**, **SearchCount**, **Reserved**)

### Parameters

**FileName** (*PSZ*) — input

Address of the ASCIIZ path name of the file or subdirectory to be found. The name component may contain global file name characters.

**DirHandle** (*PHDIR*) — input/output

Address of the handle associated with a specific DosFindFirst request. The values that can be specified are:

Value	Definition
0001H	Assign handle 1, which is always available to a process.
FFFFH	The system allocates and returns a handle.

If on input FFFFH is specified, on output DirHandle contains the handle allocated by the system.

The DosFindFirst handle is used with subsequent DosFindNext requests. Reuse of this handle in another DosFindFirst closes the association with the previous DosFindFirst and opens a new association.

**Attribute** (*USHORT*) — input

Attribute value that determines the file objects to be searched for.

Bit	Description
15 – 6	Reserved and must be zero.
5	File archive
4	Subdirectory
3	Reserved and must be zero.
2	System file
1	Hidden file
0	Read only file

These bits may be set individually or in combination. For example, an attribute value of 0021H (bits 5 and 0 set to 1) indicates a read-only file that should be archived.

If Attribute is 0, normal file entries are found. Entries for subdirectories, hidden, and system files, are not returned. If Attribute is set for hidden or system files, or directory entries, it is considered an inclusive search. All normal file entries plus all entries matching the specific attributes are returned. Set Attribute to hidden+system+directory (all three bits on) to look at all directory entries except the volume label.

Attribute cannot specify the volume label. Volume labels are queried using DosQFSInfo. Attributes of a file are queried and set with DosQFileMode and DosSetFileMode.

**ResultBuf** (*PFILEFINDBUF*) — output

Address of the structure that contains the returned directory information. The information reflects the last DosClose, DosBufReset, DosSetFileMode, DosSetFileInfo, and DosSetPathInfo calls.

**filedate** (*FDATE*)

Structure containing the date of file creation.

Bit	Description
15 – 9	Year, in binary, of file creation
8 – 5	Month, in binary, of file creation
4 – 0	Day, in binary, of file creation.

## DosFindFirst – Find First Matching File Object

**filetime (FTIME)**

Structure containing the time of file creation.

Bit	Description
15 – 11	Hours, in binary, of file creation
10 – 5	Minutes, in binary, of file creation
4 – 0	Seconds, in binary number of two-second increments, of file creation.

**fileaccessdate (FDATE)**

Structure containing the date of last access. See *FDATE* in *filedate*.

**fileaccesstime (FTIME)**

Structure containing the time of last access. See *FTIME* in *filetime*.

**writeaccessdate (FDATE)**

Structure containing the date of last write. See *FDATE* in *filedate*.

**writeaccesstime (FTIME)**

Structure containing the time of last write. See *FTIME* in *filetime*.

**filesize (ULONG)**

File size.

**filealloc (ULONG)**

Allocated file size.

**fileattrib (USHORT)**

Attributes of the file, defined in *DosSetFileMode*.

**length (UCHAR)**

Length of the ASCIIZ name string.

**matchfilename (CHAR)**

ASCIIZ name string for the first occurrence of *FileName*.

**ResultBufLen (USHORT) – input**

Length of *ResultBuf*

**SearchCount (PUSHORT) – input/output**

Address of the number of matching entries requested in *ResultBuf*. On return, this field contains the number of entries placed into *ResultBuf*.

**Reserved (ULONG) – input**

Reserved, must be set to zero.

**rc (USHORT) – return**

Return code descriptions are:

0	NO_ERROR
2	ERROR_FILE_NOT_FOUND
3	ERROR_PATH_NOT_FOUND
6	ERROR_INVALID_HANDLE
18	ERROR_NO_MORE_FILES
26	ERROR_NOT_DOS_DISK
87	ERROR_INVALID_PARAMETER
108	ERROR_DRIVE_LOCKED
111	ERROR_BUFFER_OVERFLOW
113	ERROR_NO_MORE_SEARCH_HANDLES
206	ERROR_FILENAME_EXCED_RANGE

# DosFindFirst —

## Find First Matching File Object

FAP1

### Remarks

DosFindFirst returns directory entries (up to the number requested in SearchCount) for as many files or subdirectories whose names and attributes match the specification and whose information fits in ResultBuf. On output, SearchCount contains the actual number of directory entries returned.

DosFindNext uses the directory handle associated with DosFindFirst to continue the search started by the DosFindFirst request.

Any non-zero return code indicates no handle has been allocated, including such non-error indicators as ERROR\_NO\_MORE\_FILES.

DosQSysInfo is called by an application to determine the maximum path length allowed by OS/2.

For programs running without the NEWFILES bit set, only 8.3 filename format names are returned. These names are changed to uppercase.

### Family API Considerations

Some options operate differently in the DOS mode than in OS/2 mode. Therefore, the following restrictions apply to DosFindFirst when coding for the DOS mode:

DirHandle must always equal hex 0001H or FFFFH on the initial call to DosFindFirst. Subsequent calls to DosFindFirst must have a DirHandle of hex 0001H unless a DosFindClose had been issued. In this case, 0001H or FFFFH is allowed.

### C Language

```
typedef struct _FDATE {    /* fdate */

    unsigned day   : 5;    /* binary day for directory entry */
    unsigned month : 4;    /* binary month for directory entry */
    unsigned year  : 7;    /* binary year for directory entry */

} FDATE;

typedef struct _FTIME {    /* ftime */

    unsigned twosecs : 5;    /* binary number of two-second increments */
    unsigned minutes : 6;    /* binary number of minutes */
    unsigned hours   : 5;    /* binary number of hours */

} FTIME;

typedef struct _FILEFINDBUF {    /* findbuf */

    FDATE  fdateCreation;    /* file date of creation */
    FTIME  ftimeCreation;    /* file time of creation */
    FDATE  fdateLastAccess;  /* file date of last access */
    FTIME  ftimeLastAccess;  /* file time of last access */
    FDATE  fdateLastWrite;   /* file date of last write */
    FTIME  ftimeLastWrite;   /* file time of last write */
    ULONG  cbFile;          /* file end of data */
    ULONG  cbFileAlloc;     /* file allocation */
    USHORT attrFile;        /* file attribute */
    UCHAR  cchName;         /* length of ASCIIZ name string */
    CHAR   achName[CCHMAXPATHCOMP]; /* ASCIIZ name string */

} FILEFINDBUF;
```

```
#define INCL_DOSFILEMGR
```

```
USHORT rc = DosFindFirst(FileName, DirHandle, Attribute, ResultBuf,  
                        ResultBufLen, SearchCount, Reserved);
```

```
PSZ      FileName;      /* File path name */  
PHDIR    DirHandle;     /* Directory search handle */  
USHORT   Attribute;     /* Search attribute */  
PFILEFINDBUF ResultBuf; /* Result buffer */  
USHORT   ResultBufLen;  /* Result buffer length */  
PUSHORT  SearchCount;   /* Number of entries to find */  
ULONG    0;             /* Reserved (must be zero) */  
  
USHORT   rc;            /* return code */
```

## Assembler Language

```
FDATE struc
```

```
    fdate_fs dw ?
```

```
FDATE ends
```

```
FTIME struc
```

```
    ftime_fs dw ?
```

```
FTIME ends
```

```
FILEFINDBUF struc
```

```
    findbuf_fdateCreation dw (size FDATE)/2 dup (?) ;file date of creation  
    findbuf_ftimeCreation dw (size FTIME)/2 dup (?) ;file time of creation  
    findbuf_fdateLastAccess dw (size FDATE)/2 dup (?) ;file date of last access  
    findbuf_ftimeLastAccess dw (size FTIME)/2 dup (?) ;file time of last access  
    findbuf_fdateLastWrite dw (size FDATE)/2 dup (?) ;file date of last write  
    findbuf_ftimeLastWrite dw (size FTIME)/2 dup (?) ;file time of last write  
    findbuf_cbFile dd ? ;file end of data  
    findbuf_cbFileAlloc dd ? ;file allocation  
    findbuf_attrFile dw ? ;file attribute  
    findbuf_cchName db ? ;length of ASCII name string  
    findbuf_achName db CCHMAXPATHCOMP dup (?) ;length of ASCII name string
```

```
FILEFINDBUF ends
```

```
EXTRN DosFindFirst:FAR  
INCL_DOSFILEMGR EQU 1
```

```
PUSH@ ASCIIZ FileName ;File path name string  
PUSH@ WORD DirHandle ;Directory search handle (returned)  
PUSH WORD Attribute ;Search attribute  
PUSH@ OTHER ResultBuf ;Result buffer  
PUSH WORD ResultBufLen ;Result buffer length  
PUSH@ WORD SearchCount ;Number of entries to find  
PUSH DWORD 0 ;Reserved (must be zero)  
CALL DosFindFirst
```

```
Returns WORD
```



# DosFindFirst — Find First Matching File Object

FAP1

## Example

This example searches for a file matching the pattern 'te???.dat.'

```
#define INCL_DOSFILEMGR

#define NORMAL_FILES 0
#define HIDDEN_FILES 2
#define SEARCH_PATTERN "te???.dat"
#define FILE_ATTRIBUTE NORMAL_FILES | HIDDEN_FILES
#define RESERVED 0L

HDIR      FindHandle;
FILEFINDBUF FindBuffer;
USHORT     FindCount;
USHORT     rc;

FindHandle = 0x0001;
FindCount = 1;

rc = DosFindFirst(SEARCH_PATTERN,      /* File pattern */
                  &FindHandle,        /* Directory search handle */
                  FILE_ATTRIBUTE,      /* Search attribute */
                  &FindBuffer,        /* Result buffer */
                  sizeof(FindBuffer), /* Result buffer length */
                  &FindCount,         /* # of entries to find */
                  RESERVED);           /* Reserved (must be zero) */
```

## DosFindFirst2 –

# Find First Matching File Object with Extended Attributes

This call finds the first file object or group of file objects whose name(s) match the specification. The specification can include extended attribute information associated with a file or subdirectory.

**DosFindFirst2** (**FileName**, **DirHandle**, **Attribute**, **ResultBuf**, **ResultBufLen**, **SearchCount**, **FileInfoLevel**, **Reserved**)

### Parameters

**FileName** (*PSZ*) – input

Address of the ASCIIZ path name of the file or subdirectory to be found. The name component may contain global file name characters.

**DirHandle** (*PHDIR*) – input/output

Address of the handle associated with a specific DosFindFirst2 request. The values that can be specified are:

- 0001H**      The system assigns the handle for standard output, which is always available to a process.
- FFFFH**      The system allocates and returns a handle.

If on input FFFFH is specified, on output DirHandle contains the handle allocated by the system.

The DosFindFirst2 handle is used with subsequent DosFindNext requests. Reuse of this handle in another DosFindFirst2 closes the association with the previous DosFindFirst2 and opens a new association.

**Attribute** (*USHORT*) – input

Attribute value that determines the file objects to be searched for.

Bit	Description
15 – 6	Reserved and must be zero.
5	File archive
4	Subdirectory
3	Reserved and must be zero.
2	System file
1	Hidden file
0	Read only file

These bits may be set individually or in combination. For example, an attribute value of 0021H (bits 5 and 0 set to 1) indicates a read-only file that should be archived.

To look at all directory entries except the volume label, set Attribute to hidden+system+directory (all three bits on). Attribute cannot specify the volume label. Volume labels are queried using DosQFSInfo.

If Attribute is 0, only normal file entries are found. Entries for subdirectories, hidden, and system files, are not returned.

**ResultBuf** (*PVOID*) – input/output

Address of the directory search structures for file object information levels 1 through 3. The structure required for ResultBuf is dependent on the value specified for FileInfoLevel. The information returned reflects the last DosSetFileInfo, DosSetPathInfo, DosClose, and DosBufReset calls.

#### Level 1 Information

ResultBuf contains the following structure, to which directory entry information is returned:

##### filedate (*FDATE*)

Structure containing the date of file creation.

Bit	Description
15 – 9	Year, in binary, of file creation
8 – 5	Month, in binary, of file creation

## Find First Matching File Object with Extended Attributes

4 — 0 Day, in binary, of file creation.

**filetime (FTIME)**

Structure containing the time of file creation.

Bit	Description
15 — 11	Hours, in binary, of file creation
10 — 5	Minutes, in binary, of file creation
4 — 0	Seconds, in binary number of two-second increments, of file creation.

**fileaccessdate (FDATE)**

Structure containing the date of last access. See *FDATE* in *filedate*.

**fileaccesstime (FTIME)**

Structure containing the time of last access. See *FTIME* in *filetime*.

**writeaccessdate (FDATE)**

Structure containing the date of last write. See *FDATE* in *filedate*.

**writeaccesstime (FTIME)**

Structure containing the time of last write. See *FTIME* in *filetime*.

**filesize (ULONG)**

File size.

**filealloc (ULONG)**

Allocated file size.

**fileattrib (USHORT)**

Attributes of the file, defined in *DosSetFileMode*.

**length (UCHAR)**

Length of the ASCIIZ name string.

**matchfilename (CHAR)**

ASCIIZ name string for the first occurrence of *FileName*.

**Level 2 Information**

*ResultBuf* contains a structure similar to the Level 1 structure, with the addition of the *cbList* field inserted before the name length field of the matched file object.

**cbList (ULONG)**

On output, this field contains the length of the entire EA set for the file object. This value can be used to calculate the size of the buffer required to hold EA information returned when *FileInfoLevel* = 3 is specified.

**Level 3 Information**

*ResultBuf* contains an EAOP structure, which has the following format:

**fpGEAList (PGEALIST)**

Address of GEAList. GEAList is a packed array of variable length "get EA" structures, each containing an EA name and the length of the name.

**fpFEAList (PFEALIST)**

Address of FEAList. FEAList is a packed array of variable length "full EA" structures, each containing an EA name and its corresponding value, as well as the lengths of the name and the value.

**oError (ULONG)**

Offset into structure where error has occurred.

On input, *fpGEAList* contains the address of a GEA list, which defines the attribute names whose values are to be returned. *fpGEAList* is ignored. In case of error, *oError* contains the offset of the offending GEA entry. Following is the format of the GEAList structure:

**cbList (ULONG)**

Length of the GEA list, including the length itself.

## Find First Matching File Object with Extended Attributes

### **lList (GEA)**

List of GEA structures. A GEA structure has the following format:

#### **cbName (BYTE)**

Length of EA ASCIIZ name, which does not include the null character.

#### **szName (CHAR)**

ASCIIZ name of EA.

Following the EAOP structure is a structure similar to the Level 1 structure, with the addition of an FEAList structure inserted before the name length field for the matched file object.

On output, this structure contains a packed set of records representing the directory entry and associated EAs for the matched file object. Following is the format of the FEAList structure:

### **cbList (ULONG)**

Length of the FEA list, including the length itself.

### **lList (FEA)**

List of FEA structures. An FEA structure has the following format:

#### **Flags (BYTE)**

Bit indicator describing the characteristics of the EA being defined.

<b>Bit</b>	<b>Description</b>
<b>15</b>	Critical EA.
<b>14 – 0</b>	Reserved and must be set to zero.

If bit 15 is set to 1, this indicates a critical EA. If bit 15 is 0, this means the EA is noncritical; that is, the EA is not essential to the intended use by an application of the file with which it is associated.

#### **cbName (BYTE)**

Length of EA ASCIIZ name, which does not include the null character.

#### **cbValue (USHORT)**

Length of EA value, which cannot exceed 64KB.

#### **szName (PSZ)**

ASCIIZ name of EA.

#### **aValue (PSZ)**

Free-format value of EA.

**Note:** The szName and aValue fields are not included as part of header or include files. Because of their variable lengths, these entries must be built manually.

### **ResultBufLen (USHORT) — input**

Length of ResultBuf.

### **SearchCount (USHORT) — input/output**

Address of the number of matching entries requested in ResultBuf. On return, this field contains the number of entries placed into ResultBuf.

### **FileInfoLevel (USHORT)**

The level of file information required. A value of 1, 2, or 3 can be specified. The structures described in ResultBuf indicate the information returned for each of these levels.

Regardless of the level specified, a DosFindFirst2 request (and an associated DosFindNext request) always includes the information returned by level 1 as part of the information that is returned.

However, when level 1 information is specifically requested, an Inclusive search is made. That is, all normal file entries plus all entries matching any specified attributes are returned.

### **Reserved (ULONG) — input**

Reserved, must be set to zero.

## Find First Matching File Object with Extended Attributes

**rc** (*USHORT*) — return

Return code descriptions are:

0	NO_ERROR
2	ERROR_FILE_NOT_FOUND
3	ERROR_PATH_NOT_FOUND
6	ERROR_INVALID_HANDLE
18	ERROR_NO_MORE_FILES
26	ERROR_NOT_DOS_DISK
87	ERROR_INVALID_PARAMETER
108	ERROR_DRIVE_LOCKED
111	ERROR_BUFFER_OVERFLOW
113	ERROR_NO_MORE_SEARCH_HANDLES
206	ERROR_FILENAME_EXCED_RANGE
208	ERROR_META_EXPANSION_TOO_LONG
254	ERROR_INVALID_EA_NAME
255	ERROR_EA_LIST_INCONSISTENT
275	ERROR_EAS_DIDNT_FIT

### Remarks

DosFindFirst2 returns directory entries (up to the number requested in SearchCount) and extended attribute information for as many files or subdirectories whose names, attributes, and extended attributes match the specification, and whose information fits in ResultBuf. On output, SearchCount contains the actual number of directory entries returned.

DosFindNext uses the directory handle associated with DosFindFirst2 to continue the search started by the DosFindFirst2 request.

Any non-zero return code except ERROR\_EAS\_DIDNT\_FIT indicates no handle has been allocated. This includes such non-error indicators as ERROR\_NO\_MORE\_FILES.

For programs running without the NEWFILES bit set, only 8.3 filename format names are returned. These names are changed to uppercase.

In the case of ERROR\_EAS\_DIDNT\_FIT, a search handle is returned, and a subsequent call to DosFindNext will get the next matching entry in the directory. You may use DosQPathInfo to retrieve the EAs for the matching entry by using the EA arguments that were used for the DosFindFirst2 call and the name that was returned by DosFindFirst2.

In the case of ERROR\_EAS\_DIDNT\_FIT, only information for the first matching entry is returned. This entry is the one whose EAs did not fit in the buffer. The information returned is in the format of that returned for InfoLevel 2. No further entries are returned in the buffer even if they could fit in the remaining space.

### Family API Considerations

Some options operate differently in the DOS mode than in OS/2 mode. Therefore, the following restrictions apply to DosFindFirst when coding for the DOS mode:

DirHandle must always equal hex 0001H or FFFFH on the initial call to DosFindFirst. Subsequent calls to DosFindFirst must have a DirHandle of hex 0001H unless a DosFindClose had been issued. In this case, 0001H or FFFFH is allowed.

## Find First Matching File Object with Extended Attributes

### C Language

```
typedef struct _FDATE { /* fdate */

    unsigned day : 5; /* binary day for directory entry */
    unsigned month : 4; /* binary month for directory entry */
    unsigned year : 7; /* binary year for directory entry */

} FDATE;

typedef struct _FTIME { /* ftime */

    unsigned twosecs : 5; /* binary number of two-second increments */
    unsigned minutes : 6; /* binary number of minutes */
    unsigned hours : 5; /* binary number of hours */

} FTIME;

typedef struct _FILEFINDBUF { /* findbuf */

    FDATE fdateCreation; /* file date of creation */
    FTIME ftimeCreation; /* file time of creation */
    FDATE fdateLastAccess; /* file date of last access */
    FTIME ftimeLastAccess; /* file time of last access */
    FDATE fdateLastWrite; /* file date of last write */
    FTIME ftimeLastWrite; /* file time of last write */
    ULONG cbFile; /* file end of data */
    ULONG cbFileAlloc; /* file allocation */
    USHORT attrFile; /* file attribute */
    UCHAR cchName; /* length of ASCIIZ name string */
    CHAR achName[CCHMAXPATHCOMP]; /* ASCIIZ name string */

} FILEFINDBUF;

typedef struct _FILEFINDBUF2 { /* findbuf */

    FDATE fdateCreation; /* file date of creation */
    FTIME ftimeCreation; /* file time of creation */
    FDATE fdateLastAccess; /* file date of last access */
    FTIME ftimeLastAccess; /* file time of last access */
    FDATE fdateLastWrite; /* file date of last write */
    FTIME ftimeLastWrite; /* file time of last write */
    ULONG cbFile; /* file end of data */
    ULONG cbFileAlloc; /* file allocation */
    USHORT attrFile; /* file attribute */
    ULONG cbList; /* level 2 only field (calculate size of buffer) */
    UCHAR cchName; /* length of ASCIIZ name string */
    CHAR achName[CCHMAXPATHCOMP]; /* ASCIIZ name string */

} FILEFINDBUF2;
```

# Find First Matching File Object with Extended Attributes

```

typedef struct _GEA {      /* gea */

    BYTE cbName;           /* name length not including NULL */
    CHAR szName[1];        /* attribute name */

} GEA;

typedef struct _GEALIST {  /* geal */

    ULONG cbList;          /* total bytes of structure including full list */
    GEA list[1];           /* variable length GEA structures */

} GEALIST;

typedef struct _FEA {      /* fea */

    BYTE fEA;              /* flags */
    BYTE cbName;           /* name length not including NULL */
    USHORT cbValue;        /* value length */

} FEA;

typedef struct _FEALIST {  /* feal */

    ULONG cbList;          /* total bytes of structure including full list */
    FEA list[1];           /* variable length FEA structures */

} FEALIST;

typedef struct _EAOP {     /* eaop */

    PGEALIST fpGEAList;     /* general EA list */
    PFEALIST fpFEAList;     /* full EA list */
    ULONG oError;

} EAOP;

#define INCL_DOSFILEMGR

USHORT rc = DosFindFirst2(FileName, DirHandle, Attribute, ResultBuf,
                          ResultBufLen, SearchCount, Reserved);

PSZ      FileName;         /* File path name */
PHDIR    DirHandle;        /* Directory search handle */
USHORT   Attribute;        /* Search attribute */
PVOID     ResultBuf;       /* Result buffer */
USHORT   ResultBufLen;     /* Result buffer length */
PUSHORT   SearchCount;     /* Number of entries to find */
USHORT   FileInfoLevel;    /* File data required */
ULONG     0;               /* Reserved (must be zero) */

USHORT    rc;              /* return code */

```

## Find First Matching File Object with Extended Attributes

### Assembler Language

```

FDATE    struc

    fdate_fs dw ?

FDATE    ends

FTIME    struc

    ftime_fs dw ?

FTIME    ends

FILEFINDBUF struc

    findbuf_fdateCreation dw (size FDATE)/2 dup (?) ;file date of creation
    findbuf_ftimeCreation dw (size FTIME)/2 dup (?) ;file time of creation
    findbuf_fdateLastAccess dw (size FDATE)/2 dup (?) ;file date of last access
    findbuf_ftimeLastAccess dw (size FTIME)/2 dup (?) ;file time of last access
    findbuf_fdateLastWrite dw (size FDATE)/2 dup (?) ;file date of last write
    findbuf_ftimeLastWrite dw (size FTIME)/2 dup (?) ;file time of last write
    findbuf_cbFile dd ? ;file end of data
    findbuf_cbFileAlloc dd ? ;file allocation
    findbuf_attrFile dw ? ;file attribute
    findbuf_cchName db ? ;length of ASCIIZ name string
    findbuf_achName db CCHMAXPATHCOMP dup (?) ;length of ASCIIZ name string

FILEFINDBUF ends

FILEFINDBUF2 struc

    findbuf_fdateCreation dw (size FDATE)/2 dup (?) ;file date of creation
    findbuf_ftimeCreation dw (size FTIME)/2 dup (?) ;file time of creation
    findbuf_fdateLastAccess dw (size FDATE)/2 dup (?) ;file date of last access
    findbuf_ftimeLastAccess dw (size FTIME)/2 dup (?) ;file time of last access
    findbuf_fdateLastWrite dw (size FDATE)/2 dup (?) ;file date of last write
    findbuf_ftimeLastWrite dw (size FTIME)/2 dup (?) ;file time of last write
    findbuf_cbFile dd ? ;file end of data
    findbuf_cbFileAlloc dd ? ;file allocation
    findbuf_attrFile dw ? ;file attribute
    findbuf2_cbList dd ? ;level 2 only field (calculate size of buffer)
    findbuf_achName db CCHMAXPATHCOMP dup (?) ;length of ASCIIZ name string
    findbuf_achName db 13 dup (?) ;ASCIIZ name string

FILEFINDBUF2 ends

```



# Find First Matching File Object with Extended Attributes

```

GEA    struc

    gea_cbName    db ?           ;name length not including NULL
    gea_szName    db 1 dup (?)  ;attribute name

GEA    ends

GEALIST    struc

    geal_cbList    dd ?           ;total bytes of structure including full list
    geal_list      db size GEA * 1 dup (?) ;variable length GEA structures

GEALIST    ends

FEA    struc

    fea_fEA        db ? ;flags
    fea_cbName     db ? ;name length not including NULL
    fea_cbValue    dw ? ;value length

FEA    ends

FEALIST    struc

    feal_cbList    dd ?           ;total bytes of structure including full list
    feal_list      db size FEA * 1 dup (?) ;variable length FEA structures

FEALIST    ends

EAOP    struc

    eaop_fpGEAList dd ? ;general EA list
    eaop_fpFEAList dd ? ;full EA list
    eaop_oError    dd ? ;

EAOP    ends

EXTRN DosFindFirst2:FAR
INCL_DOSFILEMGR    EQU 1

PUSH@ ASCIIZ FileName      ;File path name string
PUSH@ WORD DirHandle       ;Directory search handle (returned)
PUSH@ WORD Attribute       ;Search attribute
PUSH@ OTHER ResultBuf      ;Result buffer
PUSH@ WORD ResultBufLen    ;Result buffer length
PUSH@ WORD SearchCount     ;Number of entries to find
PUSH@ WORD FileInfoLevel   ;File data required
PUSH@ DWORD 0              ;Reserved (must be zero)
CALL DosFindFirst2

Returns WORD

```

This call locates the next set of directory entries that match the name specified in the previous `DosFindFirst`, `DosFindFirst2`, or `DosFindNext` call.

**DosFindNext** (*DirHandle*, *ResultBuf*, *ResultBufLen*, *SearchCount*)

## Parameters

**DirHandle** (*HDIR*) – input

Handle associated with a previous `DosFindFirst` or `DosFindNext` function call.

**ResultBuf** (*PFILEFINDBUF*) – output

Address of the directory search information structure. The information reflects the last `DosClose` or `DosBufReset` call.

It is possible, if the EA information for a file is 64K, that the system can never be able to return the full EA information for a file.

For the continuation of an `FileInfoLevel 3` search, this buffer should contain input in the same format as a `DosFindFirst2` `FileInfoLevel 3` search.

**filedate** (*FDATE*)

Structure containing the date of file creation.

Bit	Description
15 – 9	Year, in binary, of file creation
8 – 5	Month, in binary, of file creation
4 – 0	Day, in binary, of file creation.

**filetime** (*FTIME*)

Structure containing the time of file creation.

Bit	Description
15 – 11	Hours, in binary, of file creation
10 – 5	Minutes, in binary, of file creation
4 – 0	Seconds, in binary number of two-second increments, of file creation.

**fileaccessdate** (*FDATE*)

Structure containing the date of last access. See *FDATE* in *filedate*.

**fileaccesstime** (*FTIME*)

Structure containing the time of last access. See *FTIME* in *filetime*.

**writeaccessdate** (*FDATE*)

Structure containing the date of last write. See *FDATE* in *filedate*.

**writeaccesstime** (*FTIME*)

Structure containing the time of last write. See *FTIME* in *filetime*.

**filesize** (*ULONG*)

File size.

**filealloc** (*ULONG*)

Allocated file size.

**fileattrib** (*USHORT*)

Attributes of the file, defined in `DosSetFileMode`.

**length** (*UCHAR*)

Length of the ASCIIZ name string.

**matchfilename** (*CHAR*)

ASCIIZ name string for the first occurrence of `FileName`.

# DosFindNext — Find Next Matching File

FAPI

**ResultBufLen** (*USHORT*) — input  
Length of ResultBuf

**SearchCount** (*PUSHORT*) — input/output  
Address of the number of matching entries requested in ResultBuf. On return, this field contains the number of entries placed into ResultBuf.

**rc** (*USHORT*) — return  
Return code descriptions are:

0	NO_ERROR
6	ERROR_INVALID_HANDLE
18	ERROR_NO_MORE_FILES
26	ERROR_NOT_DOS_DISK
87	ERROR_INVALID_PARAMETER
111	ERROR_BUFFER_OVERFLOW
275	ERROR_EAS_DIDNT_FIT

## Remarks

The file name in FileName can contain global file name characters. If no more matching files are found, an error code is returned.

If an ERROR\_BUFFER\_OVERFLOW error is returned, further calls to DosFindNext will start the search from the same entry.

If an ERROR\_EAS\_DIDNT\_FIT error is returned, then the buffer was too small to hold the EAs for the first matching entry being returned. A subsequent call to DosFindNext will get the next matching entry. This enables the search to continue if the EAs being returned are too big to fit in the buffer. You may use DosQPathInfo to retrieve the EAs for the matching entry by using the EA arguments that were used for the DosFindFirst2 call and the name that was returned by DosFindFirst2.

In the case of ERROR\_EAS\_DIDNT\_FIT, only information for the first matching entry is returned. This entry is the one whose EAs did not fit in the buffer. The information returned is in the format of that returned for InfoLevel 2. No further entries are returned in the buffer even if they could fit in the remaining space.

## Family API Considerations

Some options operate differently in the DOS mode than in OS/2 mode. Therefore, the following restriction applies to DosFindNext when coding for the DOS mode:

- DirHandle must always equal 1.

## C Language

```
typedef struct _FDATE {    /* fdate */

    unsigned day   : 5;    /* binary day for directory entry */
    unsigned month : 4;    /* binary month for directory entry */
    unsigned year  : 7;    /* binary year for directory entry */

} FDATE;

typedef struct _FTIME {    /* ftime */

    unsigned twosecs : 5;    /* binary number of two-second increments */
    unsigned minutes : 6;    /* binary number of minutes */
    unsigned hours   : 5;    /* binary number of hours */

} FTIME;
```

```
typedef struct _FILEFINDBUF { /* findbuf */

    FDATE  fdateCreation;      /* file date of creation */
    FTIME  ftimeCreation;     /* file time of creation */
    FDATE  fdateLastAccess;   /* file date of last access */
    FTIME  ftimeLastAccess;   /* file time of last access */
    FDATE  fdateLastWrite;    /* file date of last write */
    FTIME  ftimeLastWrite;    /* file time of last write */
    ULONG  cbFile;            /* file end of data */
    ULONG  cbFileAlloc;       /* file allocation */
    USHORT attrFile;          /* file attribute */
    UCHAR  cchName;           /* length of ASCIIZ name string */
    CHAR   achName[13];       /* ASCIIZ name string */

} FILEFINDBUF;

#define INCL_DOSFILEMGR

USHORT rc = DosFindNext(DirHandle, ResultBuf, ResultBufLen, SearchCount);

HDIR      DirHandle; /* Directory handle */
PFILEFINDBUF ResultBuf; /* Result buffer */
USHORT     ResultBufLen; /* Result buffer length */
PUSHORT    SearchCount; /* Number of entries to find */

USHORT     rc; /* return code */
```

## Assembler Language

```
FDATE  struc

    fdate_fs dw ?

FDATE  ends

FTIME  struc

    ftime_fs dw ?

FTIME  ends

FILEFINDBUF struc

    findbuf_fdateCreation dw (size FDATE)/2 dup (?) ;file date of creation
    findbuf_ftimeCreation dw (size FTIME)/2 dup (?) ;file time of creation
    findbuf_fdateLastAccess dw (size FDATE)/2 dup (?) ;file date of last access
    findbuf_ftimeLastAccess dw (size FTIME)/2 dup (?) ;file time of last access
    findbuf_fdateLastWrite dw (size FDATE)/2 dup (?) ;file date of last write
    findbuf_ftimeLastWrite dw (size FTIME)/2 dup (?) ;file time of last write
    findbuf_cbFile dd ? ;file end of data
    findbuf_cbFileAlloc dd ? ;file allocation
    findbuf_attrFile dw ? ;file attribute
    findbuf_cchName db ? ;length of ASCIIZ name string
    findbuf_achName db 13 dup (?) ;ASCIIZ name string

FILEFINDBUF ends

EXTRN DosFindNext:FAR
INCL_DOSFILEMGR EQU 1

PUSH WORD DirHandle ;Directory search handle
PUSH@ OTHER ResultBuf ;Result buffer
PUSH WORD ResultBufLen ;Result buffer length
PUSH@ WORD SearchCount ;Number of entries to find
CALL DosFindNext

Returns WORD
```

# DosFindNext — Find Next Matching File

FAPI

## Example

This example gets the 1st file in the current directory, and then gets the next file.

```
#define INCL_DOSFILEMGR

#define NORMAL_FILES 0
#define SEARCH_PATTERN "**.*"
#define FILE_ATTRIBUTE NORMAL_FILES
#define RESERVED 0L

HDIR      FindHandle;
FILEFINDBUF FindBuffer;
USHORT     FindCount;
USHORT     rc;

FindHandle = 0x0001;
FindCount = 1;

if(!DosFindFirst(SEARCH_PATTERN,      /* File pattern */
                 &FindHandle,         /* Directory search handle */
                 FILE_ATTRIBUTE,       /* Search attribute */
                 &FindBuffer,         /* Result buffer */
                 sizeof(FindBuffer),  /* Result buffer length */
                 &FindCount,          /* # of entries to find */
                 RESERVED))           /* Reserved (must be zero) */
    rc = DosFindNext(FindHandle,      /* Directory handle */
                     &FindBuffer,    /* Result buffer */
                     sizeof(FindBuffer), /* Result buffer length */
                     &FindCount);    /* # of entries to find */
```

## DosFlagProcess – Set Process External Event Flag

This call allows one process to set an external event flag for another.

<b>DosFlagProcess</b> (ProcessID, ActionCode, Flagnum, Flagarg)
---

### Parameters

**ProcessID** (*PID*) – input

ID of the process or root process of the process tree, who is to receive notification that the flag event has occurred.

**ActionCode** (*USHORT*) – input

Indicates who is to receive notification that the flag event has occurred.

Value	Definition
0	The indicated process and all its descendant processes are flagged. (Detached processes cannot be flagged.) The indicated process must be the current process or one it has created as a non-detached process. If the indicated process terminates, its descendants are still flagged.
1	Only the indicated process is flagged. Any process can be specified.

**Flagnum** (*USHORT*) – input

Number of the flag event whose occurrence is to be signalled as shown below:

Value	Definition
0	Flag A
1	Flag B
2	Flag C.

**Flagarg** (*USHORT*) – input

An argument passed to indicated processes.

**rc** (*USHORT*) – return

Return code descriptions are:

0	NO_ERROR
1	ERROR_INVALID_FUNCTION
156	ERROR_SIGNAL_REFUSED
186	ERROR_INVALID_FLAG_NUMBER
303	ERROR_INVALID_PROCID

### Remarks

A process issues DosFlagProcess to set one of three user-defined flags available to the process. The meaning of the flag is defined by the flag user — the process that receives the signal generated by the OS/2 signal mechanism upon the setting of the flag.

When the flag user is signaled, its reaction to the signal depends on values it has specified with DosSetSigHandler. For example, it can respond by giving control to a signal handler it has registered with DosSetSigHandler for the signal that corresponds to the user flag (SIGPFA corresponds to Flag A, and so on). Or, the flag user can ignore the signal and return an error code to the flagger.

If no action is specified with DosSetSigHandler by a process for a user flag, the default is to ignore the signal.

# DosFlagProcess — Set Process External Event Flag

## C Language

```
#define INCL_DOSSIGNALS

USHORT rc = DosFlagProcess(ProcessID, ActionCode, Flagnum, Flagarg);

PID      ProcessID;    /* Process ID to flag */
USHORT   ActionCode;   /* Indicate to flag descendants */
USHORT   Flagnum;      /* Flag number */
USHORT   Flagarg;      /* Flag argument */

USHORT    rc;          /* return code */
```

## Assembler Language

```
EXTRN DosFlagProcess:FAR
INCL_DOSSIGNALS EQU 1

PUSH WORD ProcessID    ;Process ID to flag
PUSH WORD ActionCode   ;Indicate to flag descendants
PUSH WORD Flagnum      ;Flag number
PUSH WORD Flagarg      ;Flag argument
CALL DosFlagProcess
```

Returns WORD

## Example

This example starts a program named 'simple2.exe', and then signals the program with the external flag A.

```
#define INCL_DOSPROCESS
#define INCL_DOSSIGNALS

#define PROGRAM_NAME "simple2.exe"

CHAR      LoadError[100];
PSZ       Args;
PSZ       Envs;
RESULTCODES ReturnCodes;
USHORT    FlagArg;
USHORT    rc;

FlagArg = 2;

if(!DosExecPgm(LoadError,          /* Object name buffer */
               sizeof(LoadError), /* Length of object name buffer */
               EXEC_ASYNC,         /* Asynchronous/Trace flags */
               Args,               /* Argument string */
               Envs,               /* Environment string */
               &ReturnCodes,       /* Termination codes */
               PROGRAM_NAME))      /* Program file name */
    rc = DosFlagProcess(ReturnCodes.codeTerminate, /* Process ID to flag */
                        FLGP_PID,                 /* Indicate to flag descendants */
                        PFLG_A,                   /* Flag number */
                        FlagArg);                 /* Flag argument */
```

The following example illustrates the use of a user-defined flag to signal time-critical events. The main thread installs a routine, named `FlagA_Handler()`, as the signal handler for user-defined Flag A. It then creates a thread and blocks on a reserved RAM semaphore; this thread obtains its process ID and signals the main thread via Flag A. The main thread responds by executing the signal handler.

## DosFlagProcess — Set Process External Event Flag

```
#define INCL_DOSPROCESS
#define INCL_DOSSIGNALS
#define INCL_DOSERRORS

#include <os2.h>

#define TIMEOUT          5000L

TID      ThreadID;
BYTE     ThreadStack[4000];

VOID APIENTRY FlagA_Handler(arg1, arg2)      /* Define signal handler */
{
    USHORT   arg1;
    USHORT   arg2;
    {
        printf("Handler for Flag A now running.\n");
        return;
    }
}

VOID APIENTRY Thread_A()
{
    PIDINFO   PidInfo;
    USHORT    FlagArg;
    USHORT    rc;

    DosGetPID(&PidInfo);
    printf("Process ID is %d\n", PidInfo.pid);
    if(!rc = DosFlagProcess(PidInfo.pid,
                           FLGP_PID,
                           PFLG_A,
                           FlagArg))
        printf("FlagA signal sent from ThreadA to main thread.\n");
    else
        printf("FlagProcess rc is %d\n", rc)/* Error processing on rc */;
    DosExit(EXIT_THREAD,          /* Action Code */
            RETURN_CODE);         /* Result Code */
}

main()
{
    ULONG     RamSem = 0L;
    ULONG far *RamSemHandle = &RamSem;
    USHORT    rc;

    if(!rc=DosSetSigHandler((PFNSIGHANDLER) FlagA_Handler,
                           NULL,
                           NULL,
                           SIGA_ACCEPT,
                           SIG_PFLG_A))
        printf("Main thread has set FlagA handler.\n");
    else
        /* Error processing on rc */;
    if(!rc=DosSemRequest(RamSemHandle,
                        TIMEOUT)))
        printf("Semaphore obtained.\n");
    if(!(DosCreateThread((PFNTHREAD) Thread_A,
                        &ThreadID,
                        &ThreadStack[3999])))
        printf("ThreadA created.\n");
    printf("Main thread will now wait on a Ramsem for a while.\n");
    if((rc=DosSemRequest(RamSemHandle,
                        TIMEOUT))
        == ERROR_INTERRUPT)
        printf("Main thread interrupted while waiting, rc is %d.\n", rc);
}
```



# DosFreeModule — Free Dynamic Link Module

---

This call frees the reference to a dynamic link module for a process. When the dynamic link module is no longer needed by any process, the module is freed from system memory.

**DosFreeModule (ModuleHandle)**

## Parameters

**ModuleHandle (HMODULE)** — input  
Handle returned by DosLoadModule.

**rc (USHORT)** — return  
Return code descriptions are:

0	NO_ERROR
6	ERROR_INVALID_HANDLE
12	ERROR_INVALID_ACCESS
95	ERROR_INTERRUPT

## Remarks

The module identified by the handle must be loaded through DosLoadModule. An error is returned if the handle is invalid.

If any exit list routines were registered as a result of this module being loaded, the module may not be freed and ERROR\_INVALID\_ACCESS may be returned.

When DosFreeModule returns to the caller, the module handle is no longer valid and is not used to reference the dynamic link module. Procedure entry addresses returned for this module are no longer valid and cause a protection fault if they are invoked.

## C Language

```
#define INCL_DOSMODULEMGR

USHORT rc = DosFreeModule(ModuleHandle);

HMODULE      ModuleHandle; /* Module handle */

USHORT      rc;             /* return code */
```

## Assembler Language

```
EXTRN DosFreeModule:FAR
INCL_DOSMODULEMGR EQU 1

PUSH WORD ModuleHandle ;Module handle
CALL DosFreeModule

Returns WORD
```

# DosFreeModule — Free Dynamic Link Module

## Example

This example tries to load module ABCD. The system searches LIBPATH. If unsuccessful, the system tries to load the module from the program's directory (in case the user forgot to update LIBPATH).

```
#define INCL_DOSMODULEMGR

#define MODULE_NAME "abcd"
#define FULL_MODULE_NAME "\\nifty\\abcd.dll"

CHAR    LoadError[100];
HMODULE ModuleHandle;
USHORT  rc;

    if (DosLoadModule(LoadError,          /* Object name buffer */
                     sizeof(LoadError),  /* Length of object name buffer */
                     MODULE_NAME,        /* Module name string */
                     &ModuleHandle) == 2) /* Module handle */
        rc = DosLoadModule(LoadError,    /* Object name buffer */
                           sizeof(LoadError), /* Length of object name buffer */
                           FULL_MODULE_NAME, /* Module name string */
                           &ModuleHandle); /* Module handle */
    rc = DosFreeModule(ModuleHandle);      /* Module handle */
```

# DosFreeResource —

## Free Resource

---

This call frees a resource loaded by DosGetResource2.

<b>DosFreeResource (ResAddr)</b>
----------------------------------

### Parameters

**ResAddr (PBYTE)** — input

The address of the resource to free.

**rc (USHORT)** — return

Return code descriptions are:

0	NO_ERROR
6	ERROR_ACCESS_DENIED

### Remarks

DosFreeResource is used to free resources obtained with DosGetResource2.

After the last reference to a resource is freed, the memory becomes available for reuse by the system. However, the memory is not reused until the system determines it cannot satisfy a memory allocation request. This allows the resource to remain in memory in case the process issues another DosGetResource2 call. The system thus avoids having to reread the contents of the resource from disk.

### C Language

```
#define INCL_DOSFREERESOURCE

USHORT rc = DosFreeResource(ResAddr);

PBYTE      ResAddr;      /* Resource address */

USHORT      rc;           /* return code */
```

### Assembler Language

```
EXTRN DosFreeResource:FAR
INCL_DOSFREERESOURCE EQU 1

PUSH  DWORD  ResAddr      ;Resource address
CALL  DosFreeResource

Returns WORD
```

---

This call deallocates a memory segment.

<b>DosFreeSeg (Selector)</b>
------------------------------

## Parameters

**Selector (SEL)** – input

Selector of the segment to be freed.

**rc (USHORT)** – return

Return code descriptions are:

<b>0</b>	<b>NO_ERROR</b>
<b>5</b>	<b>ERROR_ACCESS_DENIED</b>
<b>212</b>	<b>ERROR_LOCKED</b>

## Remarks

DosFreeSeg frees selectors to segments returned by allocation calls to DosAllocSeg, DosAllocShrSeg, and DosAllocHuge. In addition, DosFreeSeg frees a selector returned by a call to DosCreateCSAlias. If a CS alias selector has been created for a data segment by a call to DosCreateCSAlias, the CS alias selector is still valid after the segment's data selector has been freed.

When allocated memory is shared, all selectors to the shared memory must be freed before the memory is deallocated. For example, if memory allocated by DosAllocSeg or DosAllocHuge has been given to another process with DosGiveSeg, the giver usually frees its selector by a call to DosFreeSeg. The recipient, in turn, frees the selector passed to it, after it has accessed the shared memory with DosGetSeg.

DosFreeSeg decrements the reference count for named shared segments allocated by DosAllocShrSeg. Access to the segment with DosGetShrSeg increments this count. When the count is 0, the memory is deallocated.

## Family API Considerations

Some options operate differently in the DOS mode than in OS/2 mode. Therefore, the following restriction applies to DosFreeSeg when coding for the DOS mode:

If DosFreeSeg is issued on a CSAliased segment it deallocates the associated memory. This is inconsistent with the OS/2 mode, because DosFreeSeg must be performed on both the original and CSAliased selectors.

## C Language

```
#define INCL_DOSMEMMGR
```

```
USHORT rc = DosFreeSeg(Selector);
```

```
SEL          Selector;      /* Selector */
```

```
USHORT       rc;            /* return code */
```

# DosFreeSeg — Free Segment

FAP1

## Assembler Language

```
EXTRN DosFreeSeg:FAR
INCL_DOSMEMMGR      EQU 1

PUSH  WORD  Selector      ;Selector
CALL  DosFreeSeg
```

Returns WORD

## Example

This example allocates a segment of 30,250 bytes and then discards the segment.

```
#define INCL_DOSMEMMGR

#define NUMBER_OF_BYTES 30250
#define ALLOC_FLAG SEG_GETTABLE

SEL  Selector;
USHORT rc;

rc = DosAllocSeg(NUMBER_OF_BYTES,      /* # of bytes requested */
                 &Selector,           /* Selector allocated */
                 ALLOC_FLAG);          /* Allocation flags */
rc = DosFreeSeg(Selector);             /* Segment selector */
```

# DosFSAttach – Creates and Destroys FSD Connections

DosFSAttach attaches or detaches drive to/from a remote FSD, or a pseudo-character device name to/from a local or remote FSD.

**DosFSAttach (DeviceName, FSDName, DataBuffer, DataBufferLen, OpFlag, Reserved)**

## Parameters

### **DeviceName (PSZ) – input**

Points to either the drive letter followed by a colon, or points to a pseudo-character device name. If DeviceName is a drive, it is an ASCIIZ string having the form of drive letter followed by a colon. If an attach is successful, all requests to that drive are routed to the specified FSD. If a detach is successful, the drive disappears from the system's name space.

If DeviceName is a pseudo-character device name (single file device), its format is that of an ASCIIZ string in the format of an OS/2 filename in a subdirectory called \DEV\. All requests to that name are routed to the specified FSD after a successful attach. A successful detach removes the name from the system's name space.

### **FSDName (PSZ) – input**

Address of the ASCIIZ name of the remote FSD to attach to or detach from DeviceName.

### **DataBuffer (PBYTE) – input**

Points to the user-supplied FSD argument data area. The meaning of the data is specific to the FSD. The DataBuffer contains contiguous ASCIIZ strings, with the first word of the buffer containing the number of ASCIIZ strings.

### **DataBufferLen (USHORT) – input**

The byte length of the data buffer.

### **OpFlag (USHORT) – input**

Defines the type of operation to be performed. Attach = 0 and Detach = 1.

### **Reserved (ULONG) – input**

Reserved, must be set to zero.

### **rc (USHORT) – return**

Return code descriptions are:

0	NO_ERROR
8	ERROR_NOT_ENOUGH_MEMORY
15	ERROR_INVALID_DRIVE
124	ERROR_INVALID_LEVEL
252	ERROR_INVALID_FSD_NAME
253	ERROR_INVALID_PATH

## Remarks

The redirection of drive letters representing local drives is not supported.

FSDs that wish to establish open connections that are not attached to a name in the system's name space, for such purposes as optimizing UNC connections or establishing access rights, must use an DosFSctl function to do so. DosFSAttach only creates attachments to drives or devices in the system's name space.

See description of pseudo-character devices.

# DosFSAttach — Creates and Destroys FSD Connections

## C Language

```
#define INCL_DOSFILEMGR

USHORT rc = DosFSAttach(DeviceName, FSDName, DataBuffer, DataBufferLen, OpFlag, 0);

PSZ      DeviceName;    /* Device name or drive letter string */
PSZ      FSDName;       /* FSD name */
PBYTE    DataBuffer;    /* Attach argument data */
USHORT   DataBufferLen; /* Buffer length */
USHORT   OpFlag;        /* Attach or detach */
ULONG    0;             /* Reserved (must be zero) */

USHORT   rc;            /* return code */
```

## Assembler Language

```
EXTRN DosFSAttach:FAR
INCL_DOSFILEMGR EQU 1

PUSH@ ASCIIZ DeviceName ;Device name or drive letter string
PUSH@ ASCIIZ FSDName    ;FSD name
PUSH@ OTHER DataBuffer  ;Attach argument data
PUSH WORD DataBufferLen ;Buffer length
PUSH WORD OpFlag        ;Attach or detach
PUSH DWORD 0            ;Reserved (must be zero)
CALL DosFSAttach

Returns WORD
```

# DosFSCtl – Provide File System Control

DosFSCtl provides an extended standard interface between an application and an FSD.

**DosFSCtl** (**DataArea**, **DataLengthMax**, **DataLength**, **ParmList**, **ParmLengthMax**, **ParmLength**, **FunctionCode**, **RouteName**, **FileHandle**, **RouteMethod**, **Reserved**)

## Parameters

**DataArea** (*PBYTE*) – input

Address of the data area.

**DataLengthMax** (*USHORT*) – input

The maximum length in bytes to be returned by the FSD in DataArea. DataLength may be longer than this on input, but not on output.

**DataLength** (*PUSHORT*) – input/output

Address of the length in bytes of data in DataArea. On input, it is the length of data being sent, and on output is the length of the data that was returned by the FSD. If ERROR\_BUFFER\_OVERFLOW is returned from the call, then it is the size of the buffer required to hold the data returned by the FSD.

**ParmList** (*PBYTE*) – input

Address of the command specific parameter list.

**ParmLengthMax** (*USHORT*) – input

The maximum length in bytes of the data to be returned by the FSD in ParmList. ParmLength may be longer than this on input, but not on output.

**ParmLength** (*PUSHORT*) – input/output

Address of, on input, the length in bytes of parameters passed in by the application, and on return it is the length of parameters returned by the FSD. If ERROR\_BUFFER\_OVERFLOW is returned from the call, then it is the size of the buffer required to hold the parameters returned by the FSD. No other data is returned in this case.

**FunctionCode** (*USHORT*) – input

The file system-specific function code. For remote FSDs, there are two kinds of possible FsCtl calls: FsCtl calls that are handled locally, and FsCtl calls that are exported across the network. If bit 0x4000 is set in the function code, then this indicates to the remote FSD that the function should be exported. The range of function codes are split up as follows: Function codes between 0x0000 and 0x7FFF are reserved for use by OS/2. Function codes between 0x8000 and 0xBFFF are FSD defined FsCtl functions handled by the local FSD. Function codes between 0xC000 and 0xFFFF are FSD defined FsCtl functions exported to the server.

FunctionCode = 1: returns FSD error code information. Error code is passed to the FSD in the first word of ParmList. On return, the first word of the Data area contains the length of the following ASCIIZ string. The ASCIIZ string begins at the second word and is an explanation of the error code.

**RouteName** (*PSZ*) – input

Address of the ASCIIZ name of the FSD, or the pathname of a file or directory the operation should apply to.

**FileHandle** (*HFILE*) – input

The file or device specific handle.

**RouteMethod** (*USHORT*) – input

Selects how the request is routed.

Value	Definition
-------	------------

1	FileHandle directs routing. RouteName must be a NUL pointer (0L). The FSD associated with the handle receives the request.
---	--



## DosFSctl — Provide File System Control

- 2        RouteMethod refers to a pathname, that directs routing. FileHandle must be -1. The FSD associated with the drive that the pathname refers to at the time of the request receives the request. The pathname need not refer to a file or directory that actually exists, only to a drive. A relative pathname may be used, it is processed like any other pathname.
- 3        RouteMethod refers to an FSD name, that directs routing. FileHandle must be -1. The named FSD receives the request.

**Reserved (ULONG) — input**

Reserved, must be set to zero.

**rc (USHORT) — return**

Return code descriptions are:

0	NO_ERROR
1	ERROR_INVALID_FUNCTION
6	ERROR_INVALID_HANDLE
87	ERROR_INVALID_PARAMETER
95	ERROR_INTERRUPT
111	ERROR_BUFFER_OVERFLOW
117	ERROR_INVALID_CATEGORY
124	ERROR_INVALID_LEVEL
252	ERROR_INVALID_FSD_NAME

### C Language

```
#define INCL_DOSFILEMGR
```

```
USHORT rc = DosFSctl(DataArea, DataLengthMax, DataLength, ParmList,
    ParmLengthMax, ParmLength, FunctionCode, RouteName,
    FileHandle, RouteMethod, 0);
```

```
PBYTE      DataArea;      /* Data area */
USHORT      DataLengthMax; /* Data area length */
PUSHORT     DataLength;    /* Data area length (returned) */
PBYTE      ParmList;      /* Parameter list */
USHORT      ParmLengthMax; /* Parameter list length */
PUSHORT     ParmLength;    /* Parameter list length (returned) */
USHORT      FunctionCode;  /* Function code */
PSZ         RouteName;    /* Path or FSD name */
HFILE       FileHandle;   /* File handle */
USHORT      RouteMethod;  /* Method for routing. */
ULONG       0;           /* Reserved (must be zero) */

USHORT      rc;           /* return code */
```

### Assembler Language

```
EXTRN DosFSctl:FAR
INCL_DOSFILEMGR EQU 1
```

```
PUSH@ OTHER DataArea      ;Data area
PUSH WORD DataLengthMax   ;Data area length
PUSH@ WORD DataLength     ;Data area length (returned)
PUSH@ OTHER ParmList      ;Parameter list
PUSH WORD ParmLengthMax   ;Parameter list length
PUSH@ WORD ParmLength     ;Parameter list length (returned)
PUSH WORD FunctionCode    ;Function code
PUSH@ ASCIIZ RouteName    ;Path or FSD name string
PUSH WORD FileHandle      ;File handle
PUSH WORD RouteMethod     ;Method for routing.
PUSH DWORD 0              ;Reserved (must be zero)
CALL DosFSctl
```

Returns WORD

# DosFSRamSemClear – Release a Fast-Safe RAM Semaphore

This call releases ownership of a Fast-Safe (FS) RAM semaphore.

**DosFSRamSemClear (FSRamSemStructure)**

## Parameters

**FSRamSemStructure** (*PDOSFSRSEM*) – input

Address of the FS RAM Semaphore data structure. The content/format of this structure is shown in “DosFSRamSemRequest – Request Safe RAM Semaphore” on page 2-117.

**rc** (*USHORT*) – return

Return code description is:

0                    NO\_ERROR

## Remarks

DosFSRamSemClear decrements *fs\_UseCount*, which is incremented by DosFSRamSemRequest. Recursive requests for FS RAM semaphores are supported by this use count, which keeps track of the number of times the owner has issued a DosFSRamSemRequest without a corresponding DosFSRamSemClear. If the call to DosFSRamSemClear decrements the use count to zero, the semaphore is set unowned, and any threads that were blocked waiting for the semaphore resume execution.

DosFSRamSemClear cannot be issued against a FS RAM semaphore that is owned by another thread.

## C Language

```
typedef struct _DOSFSRSEM {    /* dosfsrs */

    USHORT cb;                    /* Length of this structure (bytes) */
    PID    pid;                   /* Process ID of owner or zero */
    TID    tid;                   /* Thread ID of owner or zero */
    USHORT cUsage;                /* Reference count */
    USHORT client;                /* 16 bit field for use by owner */
    ULONG    sem;                /* OS/2 Ram Semaphore */

} DOSFSRSEM;

#define INCL_DOSSEMAPHORES

USHORT    rc = DosFSRamSemClear(FSRamSemStructure);

PDOSFSRSEM    FSRamSemStructure; /* Address of structure */

USHORT    rc;                   /* return code */
```

# DosFSRamSemClear — Release a Fast-Safe RAM Semaphore

## Assembler Language

```
DOSFSRSEM struc

    dosfsrs_cb      dw ? ;length of this structure (bytes)
    dosfsrs_pid     dw ? ;Process ID of owner or zero
    dosfsrs_tid     dw ? ;Thread ID of owner or zero
    dosfsrs_cUsage  dw ? ;reference count
    dosfsrs_client  dw ? ;16 bit field for use by owner
    dosfsrs_sem     dd ? ;OS/2 Ram Semaphore

DOSFSRSEM ends

EXTRN DosFSRamSemClear:FAR
INCL_DOSSEMAPHORES EQU 1

PUSH@ OTHER   FSRamSemStructure ;FS Ram Semaphore data structure
CALL  DosFSRamSemClear

Returns WORD
```

## Example

This example requests a FS RAM semaphore and then clears it.

```
#define INCL_DOSSEMAPHORES

#define NOT_OWNED 0
#define START 0
#define START_LONG 0L
#define TIME_OUT 1000L

DOSFSRSEM SemStruct;
USHORT rc;

SemStruct.cb = sizeof(SemStruct); /* Initialize FS Sem */
SemStruct.pid = NOT_OWNED;
SemStruct.tid = NOT_OWNED;
SemStruct.cUsage = START;
SemStruct.client = START;
SemStruct.sem = START_LONG;

if(!DosFSRamSemRequest(&SemStruct, /* Address of structure */
                      TIME_OUT)) /* Timeout */
    rc = DosFSRamSemClear(&SemStruct); /* Address of structure */
```

# DosFSRamSemRequest – Request Safe RAM Semaphore

This call obtains a Fast-Safe (FS) RAM semaphore and records the current owner for potential cleanup by a DosExitList routine.

**DosFSRamSemRequest (FSRamSemStructure,Timeout)**

## Parameters

**FSRamSemStructure** (*PDOSFSRSEM*) – input

Address of the FS RAM Semaphore data structure. The content of this structure is:

**fs\_Length** (*USHORT*)

Length in bytes of the FSRamSemStructure; 14 is the only valid value.

**fs\_ProcID** (*PID*)

Owning process ID; 0 means the semaphore is not owned.

**fs\_ThrdID** (*TID*)

Owning thread ID; 0 means the semaphore is not owned.

**fs\_UseCount** (*USHORT*)

Use count. The number of times the owning thread has issued DosFSRamSemRequest without issuing a corresponding DosFSRamSemClear.

**fs\_Client** (*USHORT*)

Is a 16-bit pattern used by the owner of a semaphore to record maintenance information about the resource managed by the semaphore.

**fs\_RAMSem** (*ULONG*)

The RAM semaphore data structure used in this request.

Before the initial call to DosFSRamSemRequest, this entire structure must be initialized to zero and its length set to 14. Other than fs\_Client, the caller should not modify any fields in this structure.

**Timeout** (*LONG*) – input

The number of milliseconds to wait for the semaphore to be cleared before resuming execution. The meaning of the specified values are:

Value	Definition
-1	The requestor waits indefinitely when the semaphore is owned. There is no time out.
0	There is an immediate return if the semaphore is owned.
>0	The value is the number of milliseconds to wait, if the semaphore is owned.

**rc** (*USHORT*) – return

Return code descriptions are:

0	NO_ERROR
121	ERROR_SEM_TIMEOUT

## Remarks

When DosFSRamSemRequest is called, it checks the status of the semaphore. If it is unowned, then DosFSRamSemRequest sets it owned, increments fs\_UseCount, and returns immediately to the caller.

If the semaphore is owned, the caller has the option to block until the semaphore is no longer owned. The unblocking of a DosFSRamSemRequest is "level triggered" because it does not actually return unless the semaphore remains clear until the affected thread can be redispached to claim it successfully. The Timeout parameter can be used to place an upper bound on the amount of time to block before returning, even though the semaphore remains owned.

When the thread is done with the protected resource, it calls DosFSRamSemClear. DosFSRamSemClear decrements fs\_UseCount. Recursive requests for FS RAM semaphores are supported by the use count, which keeps track of the number of times the owner has issued a DosFSRamSemRequest without a corre-

# DosFSRamSemRequest — Request Safe RAM Semaphore

sponding DosFSRamSemClear. If the call to DosFSRamSemClear decrements the use count to zero, the semaphore is set unowned, and any threads that were blocked waiting for the semaphore resume execution.

The 16-bit field `fs_Client` is not interpreted by the FS RAM semaphore calls. Instead, it provides the caller with a means of identifying the resource being accessed by the owner of the semaphore. This field is initialized to zero when a FS RAM semaphore is first acquired. The owner may place values into this field that describe the resource. These values can be used by an exit list handler to determine the appropriate cleanup action.

When a process terminates while owning a FS RAM semaphore, any routines in the exit list maintained by `DosExitList` are given control. These routines take appropriate steps to guarantee the integrity of resources owned by the process. To clean up a resource protected by a FS RAM semaphore, `DosFSRamSemRequest` is called to gain ownership of the semaphore. When issued during exit list processing, `DosFSRamSemRequest` examines the semaphore to determine if the semaphore is owned by the active process. It then forces the owning thread ID to be equal to the current thread ID and sets `fs_Count` = 1. This allows the exit list routine to be programmed without any FS RAM semaphore handling instructions. When the exit list routine has completed its operations, it restores the resource for use by others by issuing `DosFSRamSemClear`.

## C Language

```
typedef struct _DOSFSRSEM { /* dosfsrs */

    USHORT cb;           /* Length of this structure (bytes) */
    PID    pid;          /* Process ID of owner or zero */
    TID    tid;          /* Thread ID of owner or zero */
    USHORT cUsage;       /* Reference count */
    USHORT client;       /* 16 bit field for use by owner */
    ULONG  sem;          /* OS/2 Ram Semaphore */

} DOSFSRSEM;

#define INCL_DOSSEMAPHORES

USHORT rc = DosFSRamSemRequest(FSRamSemStructure, Timeout);

PDOSFSRSEM    FSRamSemStructure; /* Address of structure */
LONG          Timeout;           /* Timeout (in milliseconds) */

USHORT        rc;                /* return code */
```

## Assembler Language

```
DOSFSRSEM struc

    dosfsrs_cb      dw ? ;length of this structure (bytes)
    dosfsrs_pid     dw ? ;Process ID of owner or zero
    dosfsrs_tid     dw ? ;Thread ID of owner or zero
    dosfsrs_cUsage  dw ? ;reference count
    dosfsrs_client  dw ? ;16 bit field for use by owner
    dosfsrs_sem     dd ? ;OS/2 Ram Semaphore

DOSFSRSEM ends

EXTRN DosFSRamSemRequest:FAR
INCL_DOSSEMAPHORES EQU 1

PUSH@ OTHER    FSRamSemStructure ;FS Ram Semaphore data structure
PUSH    DWORD  Timeout           ;Timeout (in milliseconds)
CALL    DosFSRamSemRequest

Returns WORD
```

## DosFSRamSemRequest — Request Safe RAM Semaphore

### Example

This example requests a FS RAM semaphore.

```
#define INCL_DOSSEMAPHORES

#define NOT_OWNED 0
#define START 0
#define START_LONG 0L
#define TIME_OUT 1000L

DOSFSRSEM SemStruct;
USHORT rc;

SemStruct.cb = sizeof(SemStruct); /* Initialize FS Sem */
SemStruct.pid = NOT_OWNED;
SemStruct.tid = NOT_OWNED;
SemStruct.cUsage = START;
SemStruct.client = START;
SemStruct.sem = START_LONG;

rc = DosFSRamSemRequest(&SemStruct, /* Address of structure */
                       TIME_OUT); /* Timeout */
```

This call obtains a collating sequence table (for characters hex 00H through FFH) that resides in the country information file. It is used by the SORT utility to sort text according to the collating sequence.

**DosGetCollate (Length, Country, MemoryBuffer, DataLength)**

## Parameters

**Length** (*USHORT*) — input

Length, in bytes, of the data area (MemoryBuffer). A length value of 256 is sufficient.

**Country** (*PCOUNTRYCODE*) — input

Address of the country information structure:

**country** (*USHORT*)

Country code identifier. 0 is the default country code.

**codepage** (*USHORT*)

Code page identifier. 0 is the code page of the current process.

**MemoryBuffer** (*PCHAR*) — output

Address of the collating table. This memory area is provided by the caller. The size of the area is provided by the input parameter Length. If it is too small to hold all the available information then as much information as possible is provided in the available space (in the order that the data would appear). If the amount of returned data does not fill the memory area provided by the caller, the unaltered memory is set at 0. The buffer format for the returned information is:

Byte	Description
0	Sort weight of ASCII (0)
1	Sort weight of ASCII (1)
:	
255	Sort weight of ASCII (255)

**DataLength** (*PUSHORT*) — output

Address of the length, in bytes, of the collate table.

**rc** (*USHORT*) — return

Return code descriptions are:

0	NO_ERROR
396	ERROR_NLS_NO_COUNTRY_FILE
397	ERROR_NLS_OPEN_FAILED
398	ERROR_NO_COUNTRY_OR_CODEPAGE
399	ERROR_NLS_TABLE_TRUNCATED

## Remarks

The returned country-dependent information can be for the default country and current process code page or for a specific country and code page. For more information see "DosSetCp — Set Code Page" on page 2-320.

## C Language

```
typedef struct _COUNTRYCODE { /* ctryc */

    USHORT country;          /* country code */
    USHORT codepage;         /* code page */

} COUNTRYCODE;

#define INCL_DOSNLS

USHORT rc = DosGetCollate(Length, Structure, MemoryBuffer, DataLength);

USHORT      Length;          /* Length of data area provided */
PCOUNTRYCODE Structure;      /* Input data structure */
PCHAR       MemoryBuffer;    /* Collate table (returned) */
PUSHORT     DataLength;      /* Length of collate table (returned) */

USHORT      rc;              /* return code */
```

## Assembler Language

```
COUNTRYCODE struc

    ctryc_country dw ? ;country code
    ctryc_codepage dw ? ;code page

COUNTRYCODE ends

EXTRN DosGetCollate:FAR
INCL_DOSNLS EQU 1

PUSH WORD Length ;Length of data area provided
PUSH@ OTHER Structure ;Input data structure
PUSH@ OTHER MemoryBuffer ;Collate table (returned)
PUSH@ WORD DataLength ;Length of collate table (returned)
CALL DosGetCollate
```

Returns WORD

## Example

This example gets a collating sequence table for codepage 850 and the current country.

```
#define INCL_DOSNLS

#define CURRENT_COUNTRY 0
#define NLS_CODEPAGE 850

COUNTRYCODE Country;
CHAR CollBuffer[256];
USHORT Length;
USHORT rc;

Country.country = CURRENT_COUNTRY;
Country.codepage = NLS_CODEPAGE;

rc = DosGetCollate(sizeof(CollBuffer), /* Length of data area provided */
                  &Country,           /* Input data structure */
                  CollBuffer,         /* Data area to contain collate table */
                  &Length);          /* Length of table */
```



# DosGetCp —

## Get Process Code Page

FAPI

This call allows a process to query the current process code page and the prepared system code pages.

**DosGetCp** (*Length*, *CodePageList*, *DataLength*)

### Parameters

**Length** (*USHORT*) — input

Length, in bytes, of *CodePageList*. This length should be at least 2 bytes. If the length is less than the bytes needed to return all the prepared system code pages than the returned *CodePageList* is truncated.

**CodePageList** (*PUSHORT*) — output

Address of the list of available system code pages. The format of the information returned in this list is:

Word	Description
1	Current code page identifier
2	The first prepared code page
3	The second prepared code page
N	Other prepared system code pages.

**DataLength** (*PUSHORT*) — output

Address of the length, in bytes, of the returned data.

**rc** (*USHORT*) — return

Return code descriptions are:

0	NO_ERROR
473	ERROR_CPLIST_TOO_SMALL

### Remarks

If the process code page identifier was previously set by *DosSetCp* or inherited by a process, it is returned to the caller.

An input list size (*length*) of two bytes returns only the current process code page identifier. If *CodePageList* length is too small to hold all the available information, then as much information as possible is provided in the available space and *ERROR\_CPLIST\_TOO\_SMALL* is returned.

If no codepages were prepared with the *CODEPAGE* command, a length of two and current codepage identifier value of zero is returned.

### Family API Considerations

Some options operate differently in the DOS mode than in OS/2 mode. Therefore, the following restriction applies to *DosGetCp* when coding for the DOS mode:

The current process code page, and no more than one prepared system code page, is returned.

### C Language

```
#define INCL_DOSNLS

USHORT rc = DosGetCp(Length, CodePageList, DataLength);

USHORT Length;      /* Length of list */
PUSHORT CodePageList; /* List (returned) */
PUSHORT DataLength;  /* Length of list (returned) */

USHORT rc;          /* return code */
```

**Assembler Language**

```
EXTRN DosGetCp:FAR
INCL_DOSNLS      EQU 1

PUSH  WORD    Length      ;Length of list
PUSH@ WORD    CodePageList ;List (returned)
PUSH@ WORD    DataLength   ;Length of list (returned)
CALL  DosGetCp
```

Returns WORD

**Example**

This example gets the current code page and then up to 3 other prepared codepages.

```
#define INCL_DOSNLS

USHORT CpList[8];
USHORT CpSize;
USHORT rc;

    rc = DosGetCp(sizeof(CpList),      /* Length of list */
                  CpList,              /* List */
                  &CpSize);           /* Length of returned list */
```

# DosGetCtryInfo — Get Country Information

FAP1

This call obtains country-dependent formatting information that resides in the country information file.

**DosGetCtryInfo** (**Length**, **Country**, **MemoryBuffer**, **DataLength**)

## Parameters

**Length** (*USHORT*) — input

Length, in bytes, of the data area (**MemoryBuffer**). This length should be at least 38 bytes.

**Country** (*PCOUNTRYCODE*) — input

Address of the country information structure:

**country** (*USHORT*)

Country code identifier; 0 is the default country code.

**codepage** (*USHORT*)

Code page identifier; 0 is the code page of the current process.

**MemoryBuffer** (*PCOUNTRYINFO*) — output

Address of the buffer where the country-dependent data is returned. The application must request enough space in memory, a minimum of 38 bytes, to hold the returned data. If the requested space is not large enough, the system fills the allocated space with as much data as it can hold. If the requested space is too large, the extra memory requested is set to 0. The data is returned in the buffer in the following format:

**country** (*USHORT*)

Country code identifier.

**codepage** (*USHORT*)

Code page identifier.

**dateformat** (*USHORT*)

Date format.

Value	Definition
-------	------------

0	mm/dd/yy
---	----------

1	dd/mm/yy
---	----------

2	yy/mm/dd
---	----------

**currency** (*CHAR*)

Currency identifier, terminated with a null.

**thousandsep** (*CHAR*)

Thousands separator, terminated with a null.

**decimalsep** (*CHAR*)

Decimal separator, terminated with a null.

**datesep** (*CHAR*)

Date separator, terminated with a null.

**timesep** (*CHAR*)

Time separator, terminated with a null.

## DosGetCtryInfo — Get Country Information

**currencyformat (UCHAR)**

The currency bit fields in the following format:

Bit	Description
7 – 3	Reserved
2	0 = Bits 0 and 1 are used. 1 = Bits 0 and 1 are ignored; the currency indicator replaces the decimal indicator.
1	0 = No space between the currency indicator and money value. 1 = One space between the currency indicator and money value.
0	0 = Currency indicator precedes the money value. 1 = Currency indicator follows the money value.

**decimalspace (UCHAR)**

Number (in binary) of decimal spaces used to indicate currency value.

**timeformat (UCHAR)**

Time format for presentation in file directory:

Bit	Description
7 – 1	Reserved
0	0 = 12-hour format 1 = 24-hour format.

**reserved (USHORT)**

Reserved, set to zero.

**datallstsep (CHAR)**

Data list separator, terminated with a null.

**reserved (USHORT)**

Reserved, set to zero.

**DataLength (USHORT) – output**

Address of the length, in bytes, of the country dependent information.

**rc (USHORT) – return**

Return code descriptions are:

0	NO_ERROR
396	ERROR-NLS_NO_COUNTRY_FILE
397	ERROR-NLS_OPEN_FAILED
398	ERROR_NO_COUNTRY_OR_CODEPAGE
399	ERROR-NLS_TABLE_TRUNCATED

**Remarks**

The returned country dependent information may be for the default country and current process code page, or for a specific country and code page. For more information on code page, see "DosSetCp — Set Code Page" on page 2-320.

**Family API Considerations**

Some options operate differently in DOS mode than in OS/2 mode. Note that not all country information is available in DOS 3.3.

# DosGetCtryInfo — Get Country Information

FAPI

## C Language

```
typedef struct _COUNTRYCODE { /* ctryc */

    USHORT country;          /* country code */
    USHORT codepage;         /* code page */

} COUNTRYCODE;

typedef struct _COUNTRYINFO { /* ctryi */

    USHORT country;          /* country code */
    USHORT codepage;         /* code page */
    USHORT fsDateFmt;        /* date format */
    CHAR szCurrency[5];      /* currency indicator */
    CHAR szThousandsSeparator[2]; /* thousands separator */
    CHAR szDecimal[2];       /* decimal separator */
    CHAR szDateSeparator[2]; /* date separator */
    CHAR szTimeSeparator[2]; /* time separator */
    UCHAR fsCurrencyFmt;     /* bit fields for currency format */
    UCHAR cDecimalPlace;     /* currency decimal places */
    UCHAR fsTimeFmt;         /* Time format (AM/PM or 24 hr) */
    USHORT abReserved1[2];    /* reserved (0) */
    CHAR szDataSeparator[2]; /* Data list separator */
    USHORT abReserved2[5];    /* reserved (0) */

} COUNTRYINFO;

#define INCL_DOSNLS

USHORT rc = DosGetCtryInfo(Length, Structure, MemoryBuffer, DataLength);

USHORT      Length;          /* Length of data area provided */
PCOUNTRYCODE Structure;      /* Input data structure */
PCOUNTRYINFO MemoryBuffer;   /* Country information (returned) */
PUSHORT     DataLength;      /* Length of data (returned) */

USHORT      rc;              /* return code */
```

## Assembler Language

COUNTRYCODE struc

```
    ctryc_country   dw ? ;country code
    ctryc_codepage  dw ? ;code page
```

COUNTRYCODE ends

COUNTRYINFO struc

```
    ctryi_country       dw ? ;country code
    ctryi_codepage      dw ? ;code page
    ctryi_fsDateFmt     dw ? ;date format
    ctryi_szCurrency    db 5 dup (?) ;currency indicator
    ctryi_szThousandsSeparator db 2 dup (?) ;thousands separator
    ctryi_szDecimal     db 2 dup (?) ;decimal separator
    ctryi_szDateSeparator db 2 dup (?) ;date separator
    ctryi_szTimeSeparator db 2 dup (?) ;time separator
    ctryi_fsCurrencyFmt db ? ;bit fields for currency format
    ctryi_cDecimalPlace db ? ;currency decimal places
    ctryi_fsTimeFmt     db ? ;Time format (AM/PM or 24 hr)
    ctryi_abReserved1   dw 2 dup (?) ;reserved (0)
    ctryi_szDataSeparator db 2 dup (?) ;Data list separator
    ctryi_abReserved2   dw 5 dup (?) ;reserved (0)
```

COUNTRYINFO ends

```
EXTRN DosGetCtryInfo:FAR
INCL_DOSNLS      EQU 1
```

```
PUSH WORD Length ;Length of data area provided
PUSH@ OTHER Structure ;Input data structure
PUSH@ OTHER MemoryBuffer ;Country information (returned)
PUSH@ WORD DataLength ;Length of data (returned)
CALL DosGetCtryInfo
```

Returns WORD

## Example

This example gets country-dependent information.

```
#define INCL_DOSNLS
```

```
#define CURRENT_COUNTRY 0
```

```
#define NLS_CODEPAGE 850
```

```
COUNTRYCODE Country;
COUNTRYINFO CtryBuffer;
USHORT Length;
USHORT rc;
```

```
Country.country = CURRENT_COUNTRY;
Country.codepage = NLS_CODEPAGE;
```

```
rc = DosGetCtryInfo(sizeof(CtryBuffer), /* Length of data area provided */
                    &Country,          /* Input data structure */
                    &CtryBuffer,       /* Data area to be filled by function */
                    &Length);          /* Length of data returned */
```

# DosGetDateTime — Get Current Date and Time

FAPI

---

This call gets the current date and time maintained by the operating system.

<b>DosGetDateTime</b> ( <i>DateTime</i> )
---

## Parameters

**DateTime** (*PDATETIME*) — output

Address of the date and time structure:

**hours** (*UCHAR*)

Current hour.

**minutes** (*UCHAR*)

Current minute.

**seconds** (*UCHAR*)

Current second.

**hundredths** (*UCHAR*)

Current hundredth of a second.

**day** (*UCHAR*)

Current day.

**month** (*UCHAR*)

Current month.

**year** (*USHORT*)

Current year.

**timezone** (*SHORT*)

Minutes west of UTC (Universal Time Coordinate).

**weekday** (*UCHAR*)

Current day of the week. Sunday is 0.

**rc** (*USHORT*) — return

Return code description is:

0            NO\_ERROR

## Remarks

The dayofweek value is based on Sunday equal to zero. The value of timezone is the difference in minutes between the current time zone and UTC. This number is positive if it is earlier than UTC and negative if it is later than UTC. For Eastern Standard Time, this value is 300 (5 hours earlier than UTC).

If the application is executing in the OS/2 environment, it is more efficient to obtain these variables by calling DosGetInfoSeg instead of this function. However, applications written to the family API cannot depend on the availability of DosGetInfoSeg.

## C Language

```
typedef struct _DATETIME { /* date */

    UCHAR   hours;          /* current hour */
    UCHAR   minutes;        /* current minute */
    UCHAR   seconds;        /* current second */
    UCHAR   hundredths;     /* current hundredths of a second */
    UCHAR   day;            /* current day */
    UCHAR   month;          /* current month */
    USHORT  year;           /* current year */
    SHORT   timezone;       /* minutes of time west of UTC */
    UCHAR   weekday;        /* current day of week */

} DATETIME;

#define INCL_DOSDATETIME

USHORT rc = DosGetDateTime(DateTime);

PDATETIME    DateTime;    /* Address of date/time structure (returned) */

USHORT       rc;          /* return code */
```

## Assembler Language

```
DATETIME struc

    date_hours      db ? ;current hour
    date_minutes    db ? ;current minute
    date_seconds     db ? ;current second
    date_hundredths db ? ;current hundredths of a second
    date_day        db ? ;current day
    date_month       db ? ;current month
    date_year        dw ? ;current year
    date_timezone    dw ? ;minutes of time west of UTC
    date_weekday     db ? ;current day of week

DATETIME ends

EXTRN DosGetDateTime:FAR
INCL_DOSDATETIME EQU 1

PUSH@ OTHER DateTime    ;Date/time structure (returned)
CALL DosGetDateTime

Returns WORD
```

## Example

This example gets the current time and date.

```
#define INCL_DOSDATETIME

DATETIME DateBuffer;
USHORT rc;

rc = DosGetDateTime(&DateBuffer);    /* Date/Time structure */
```



## DosGetDateTime — Get Current Date and Time

FAPI

The following example obtains and prints date and time information. It then changes the system date to 5/10/1987 and prints the updated information.

```
#define INCL_DOSDATETIME

#include <os2.h>

main()
{
    DATETIME    DateTime;      /* Structure to hold date/time info. */
    USHORT      rc;

    rc = DosGetDateTime(&DateTime);    /* Address of d/t structure */
    printf("Today is %d-%d-%d; the time is %d:%d\n", DateTime.month,
        DateTime.day, DateTime.year, DateTime.hours, DateTime.minutes);
    DateTime.day = 10;
    DateTime.month = 5;
    DateTime.year = 1987;
    printf("The new date is %d-%d-%d; the time is %d:%d\n", DateTime.month,
        DateTime.day, DateTime.year, DateTime.hours, DateTime.minutes);
    rc = DosSetDateTime(&DateTime);    /* Address of d/t structure */
    printf("rc is %d\n", rc);
}
```

This call obtains a DBCS (double byte character set) environmental vector that resides in the country information file.

**DosGetDBCSEv (Length, Country, MemoryBuffer)**

## Parameters

**Length** (*USHORT*) — input

Length, in bytes, of the data area (MemoryBuffer). This value should be at least 10.

**Country** (*PCOUNTRYCODE*) — input

Address of the country information structure:

**countrycode** (*USHORT*)

Country code identifier. 0 is the default country code.

**codepage** (*USHORT*)

Code page identifier. 0 is the code page of the current process.

**MemoryBuffer** (*PCHAR*) — output

Address of the country dependent information for the DBCS environmental vector. This memory area is provided by the caller. The size of the area is provided by the input parameter Length. If it is too small to hold all the available information, then as much information as possible is provided in the available space. The format of the information returned in this buffer is:

Word	Description
1	First range definition for DBCS lead byte values
	<b>High byte</b> Binary start value (inclusive) for range one
	<b>Low byte</b> Binary stop value (inclusive) for range one.
2	Second range definition
	<b>High byte</b> Binary start value for range two
	<b>Low byte</b> Binary stop value for range two.
N	Nth range definition
	<b>High byte</b> Binary start value for Nth range
	<b>Low byte</b> Binary stop value for Nth range.
N + 1	Two bytes of binary 0 terminate list.

For example:

```
DB 81H,9FH
DB E0H,FCH
DB 00H,00H
```

**rc** (*USHORT*) — return

Return code descriptions are:

0	NO_ERROR
396	ERROR_NLS_NO_COUNTRY_FILE
397	ERROR_NLS_OPEN_FAILED
398	ERROR_NO_COUNTRY_OR_CODEPAGE
399	ERROR_NLS_TABLE_TRUNCATED

## Remarks

The returned DBCS environmental vector may be for the default country and current process code page or for a specific country and code page. For more information on code page see "DosSetCp — Set Code Page" on page 2-320.

# DosGetDBCSEv — Get DBCS Environmental Vector

FAPI

## C Language

```
typedef struct _COUNTRYCODE { /* ctryc */

    USHORT country;           /* country code */
    USHORT codepage;          /* code page */

} COUNTRYCODE;

#define INCL_DOSNLS

USHORT rc = DosGetDBCSEv(Length, Structure, MemoryBuffer);

USHORT      Length;          /* Length of data area provided */
PCOUNTRYCODE Structure;      /* Input data structure */
PCHAR       MemoryBuffer;    /* DBCS environmental vector (returned) */

USHORT      rc;              /* return code */
```

## Assembler Language

```
COUNTRYCODE struc

    ctryc_country dw ? ;country code
    ctryc_codepage dw ? ;code page

COUNTRYCODE ends

EXTRN DosGetDBCSEv:FAR
INCL_DOSNLS EQU 1

PUSH WORD Length ;Length of data area provided
PUSH@ OTHER Structure ;Input data structure
PUSH@ OTHER MemoryBuffer ;DBCS environmental vector (returned)
CALL DosGetDBCSEv

Returns WORD
```

# DosGetEnv —

## Get Address of Process Environment String

---

This call returns the address of the process environment string for the current process.

**DosGetEnv (EnvSegment, CmdOffset)**

### Parameters

**EnvSegment (PSEL)** — output

Address of the selector for the environment segment.

**CmdOffset (PUSHORT)** — output

Address of the offset to the command line within the environment segment.

**rc (USHORT)** — return

Return code descriptions are:

0	NO_ERROR
12	ERROR_INVALID_ACCESS

### Remarks

DosGetEnv can be used by dynamic link library routines that need to determine the environment for the current process.

When a process issues DosExecPgm to start another process, the program that receives control is returned a pointer to the environment segment.

### C Language

```
#define INCL_DOSMISC
```

```
USHORT rc = DosGetEnv(EnvSegment, CmdOffset);
```

```
PUSHORT EnvSegment; /* Selector (returned) */
PUSHORT CmdOffset; /* Command line offset (returned) */

USHORT rc; /* return code */
```

### Assembler Language

```
EXTRN DosGetEnv:FAR
INCL_DOSMISC EQU 1

PUSH@ WORD EnvSegment ;Selector (returned)
PUSH@ WORD CmdOffset ;Command line offset (returned)
CALL DosGetEnv

Returns WORD
```

## Get Address of Process Environment String

### Example

The following example shows how one may obtain information for program initialization. The program locates the environment segment and prints the name of the command from the command line. It then obtains the OS/2 version number and prints it.

```
#define INCL_DOS

#include <os2.h>

#define ENVVARNAME "PATH"

main()
{
    SEL      EnvSel;           /* Environment segment selector (returned) */
    USHORT   CmdOffset;        /* Offset into env. seg. of command line (returned) */
    PSZ FAR  *Commandline;     /* Pointer made by EnvSel and CmdOffset */
    USHORT   Version;          /* Version numbers (returned) */
    BYTE     MajorVer;         /* Major version number */
    BYTE     MinorVer;         /* Minor version number */
    USHORT   rc;               /* return code */

    /** Locate environment segment and offset of command line. **/
    if(!rc=DosGetEnv(&EnvSel, /* Env. seg. selector (returned) */
                    &CmdOffset)) /* Offset of command line (returned) */
        printf("Environment located; selector is %x offset is %x\n", EnvSel,
              CmdOffset);

    /** Use a macro to make a far pointer out of selector:offset pair. **/
    /** Notice the far-string pointer specification (%Fs) used to print **/

    Commandline = MAKEP(EnvSel, CmdOffset);
    printf("Command entered is %Fs.\n", Commandline);

    /** Obtain and print version info; use macros to extract info. **/
    /** We need to divide by 10 to obtain true version numbers. **/

    if(!rc=DosGetVersion(&Version))
    {
        MajorVer = HIBYTE(Version) / 10;
        MinorVer = LOBYTE(Version) / 10;
        printf("This is OS/2 version %d.%d\n", MajorVer, MinorVer);
    }
}
```

---

This call returns a shift count used to derive the selectors that address memory allocated with DosAllocHuge.

<b>DosGetHugeShift (ShiftCount)</b>
-------------------------------------

## Parameters

**ShiftCount** (*PUSHORT*) – output  
Address of the shift count.

**rc** (*USHORT*) – return  
Return code description is:

0            NO\_ERROR

## Remarks

Each segment of a huge memory allocation has a unique selector. To compute the next sequential selector in a huge memory area, take the value 1, shift it left by the number of bits specified in shift count. Use the resulting value as an increment to add to the previous selector (using the selector returned by DosAllocHuge as the first selector). For example:

- Assume DosAllocHuge is issued with NumSeg equal to 3, and that the number 63 is returned for the selector of the first segment.
- If DosGetHugeShift returns a shift count of 4, shifting the value "1" by this amount results in an increment of 16.
- Adding this increment to selector number 63 produces 79 for the second selector. Adding the same increment to selector number 79 yields 95 for the third selector.

## C Language

```
#define INCL_DOSMEMMGR

USHORT rc = DosGetHugeShift(ShiftCount);

PUSHORT        ShiftCount;    /* Shift Count (returned) */

USHORT        rc;            /* return code */
```

## Assembler Language

```
EXTRN DosGetHugeShift:FAR
INCL_DOSMEMMGR        EQU 1

PUSH@ WORD    ShiftCount    ;Shift Count (returned)
CALL    DosGetHugeShift

Returns WORD
```

# DosGetInfoSeg — Get Address of System Variables Segment

---

This call returns the address of a global and local information segment, specific to a process.

<b>DosGetInfoSeg</b> ( <b>GlobalSeg</b> , <b>LocalSeg</b> )
---

## Parameters

**GlobalSeg** (*PSEL*) — output

Address of the global information segment structure, as defined below:

**time** (*ULONG*)

Time in seconds since 1/1/1970.

**millisecs** (*ULONG*)

Time in milliseconds.

**hours** (*UCHAR*)

Current hour.

**minute** (*UCHAR*)

Current minute.

**seconds** (*UCHAR*)

Current second.

**hundredsec** (*UCHAR*)

Current hundredth of a second.

**timezone** (*USHORT*)

Minutes from UTC; if hex FFFFH, timezone is undefined.

**Interval** (*USHORT*)

Timer interval in tenths of milliseconds.

**day** (*UCHAR*)

Day.

**month** (*UCHAR*)

Month.

**year** (*USHORT*)

Year.

**weekday** (*UCHAR*)

Day-of-week:

Value	Definition
0	Sunday
1	Monday
2	Tuesday
3	Wednesday
4	Thursday
5	Friday
6	Saturday

**majorversion** (*UCHAR*)

Major version number.

**minorversion** (*UCHAR*)

Minor version number.

**revision** (*UCHAR*)

Revision letter.

## DosGetInfoSeg – Get Address of System Variables Segment

**currentsession (UCHAR)**

Current foreground full-screen session.

**maxnumsessions (UCHAR)**

Maximum number of full-screen sessions.

**hugeshift (UCHAR)**

Shift count for huge segments.

**protmodelnd (UCHAR)**

Protect-mode-only indicator:

Value	Definition
0	DOS mode and OS/2 mode.
1	OS/2 mode only.

**lastprocess (USHORT)**

Process ID of the current foreground process.

**dynvarflag (UCHAR)**

Dynamic variation flag:

Value	Definition
0	Absolute
1	Enabled.

**maxwait (UCHAR)**

Maximum wait in seconds.

**mintimeslice (USHORT)**

Minimum timeslice in milliseconds.

**maxtimeslice (USHORT)**

Maximum timeslice in milliseconds.

**bootdrive (USHORT)**

Drive from which the system was booted:

Value	Definition
1	Drive A.
2	Drive B.
:	
n	Drive n.

**traceflags (UCHAR)**

32 system trace major code flags. Each bit corresponds to a trace major code, hex 00-FFH. The most significant bit (left-most) of the first byte corresponds to major code hex 00H.

Value	Definition
0	Trace disabled.
1	Trace enabled.

**maxtextsessions (UCHAR)**

Maximum number of VIO windowable sessions.

**maxpmsessions (UCHAR)**

Maximum number of Presentation Manager sessions.

**LocalSeg (PSEL) – output**

Address of the selector for the local information segment structure, as defined below:

**processid (PID)**

Current process ID.

**parentprocessid (PID)**

Parent process ID.



## DosGetInfoSeg — Get Address of System Variables Segment

**threadprty** (*USHORT*)

Priority of current thread.

**threadId** (*TID*)

Current thread ID.

**sessionId** (*USHORT*)

Current session ID.

**procstatus** (*UCHAR*)

Process status.

**unused** (*UCHAR*)

Unused.

**foregroundprocess** (*BOOL*)

Current process is in foreground (has keyboard focus). Value —1 implies yes, 0 implies no.

**typeProcess** (*UCHAR*)

Type of process:

Value	Definition
0	Full screen protect mode session.
1	Requires real mode.
2	VIO windowable protect mode session.
3	Presentation Manager protect mode session.
4	Detached protect mode process.

**unused** (*UCHAR*)

Unused.

**environmentsel** (*SEL*)

Environment selector.

**cmdlineoff** (*USHORT*)

Command line offset in the segment addressed by environmentsel.

**dataseglen** (*USHORT*)

Length of data segment in bytes.

**stacksize** (*USHORT*)

Stack size in bytes.

**heapsize** (*USHORT*)

Heap size in bytes.

**hmodule** (*HMODULE*)

Module handle.

**dssel** (*SEL*)

Data segment selector.

**rc** (*USHORT*) — return

Return code description is:

0 NO\_ERROR

### Remarks

Items of general interest are kept in the global information segment. Items of information specific to a particular process are kept in the local information segment. This information can be directly read by the application program. Both of these segments are defined as read-only segments. The application program cannot modify this data.

Assuming  $n_1$ ,  $n_2$ , and  $n_3$  are the maximum number of full-screen sessions, VIO-windowable sessions, and Presentation Manager sessions, the first 0 through  $(n_1-1)$  session numbers are assigned to full-screen sessions. The next  $n_2$  session numbers are assigned to VIO-windowable sessions, and the next  $n_3$  session numbers are assigned to Presentation Manager sessions. Session numbers in the range  $(n_1+n_2+n_3)$  through 255 are reserved. Applications should use  $(n_1+n_2+n_3-1)$  as an upper boundary.

## DosGetInfoSeg – Get Address of System Variables Segment

Applications should not assume that all session numbers starting with (n1+n2) and higher are Presentation Manager sessions.

The application program must be careful when referencing the date or time fields in the global information segment. A timer interrupt can be received by the system in between the instructions that read the individual fields, changing the data in these fields. For example, if the time is currently 23:59:59 on Tuesday, 6/2/87, the program can read the hours field (23). A timer interrupt can now be received, changing the time to 00:00:00 on Wednesday, 6/3/87. The program reads the rest of the time field (0 minutes) and the date field. The program would then think the current time and date is 23:00:00 on Wednesday, 6/3/87, which is incorrect.

The application program should read all time and date fields it uses as quickly as possible. It can then compare the least significant time field it uses (milliseconds, hundredths, seconds, or minutes) against the current value in the global information segment. If the value has not changed, the rest of the information is valid. If the value has changed, the program time or date information should be read again, as the information is updated while the program reads it.

### C Language

```
typedef struct _GINFOSEG {  
  
    ULONG    time;           /* time in seconds */  
    ULONG    msec;          /* milliseconds */  
    UCHAR    hour;          /* hours */  
    UCHAR    minutes;       /* minutes */  
    UCHAR    seconds;       /* seconds */  
    UCHAR    hundredths;    /* hundredths */  
    USHORT   timezone;      /* minutes from UTC */  
    USHORT   cusecTimerInterval; /* timer interval (units = 0.0001 seconds) */  
    UCHAR    day;           /* day */  
    UCHAR    month;         /* month */  
    USHORT   year;          /* year */  
    UCHAR    weekday;       /* day of week */  
    UCHAR    uchMajorVersion; /* major version number */  
    UCHAR    uchMinorVersion; /* minor version number */  
    UCHAR    chRevisionLetter; /* revision letter */  
    UCHAR    sgCurrent;     /* current foreground session */  
    UCHAR    sgMax;         /* maximum number of sessions */  
    UCHAR    cHugeShift;    /* shift count for huge elements */  
    UCHAR    fProtectModeOnly; /* protect mode only indicator */  
    USHORT   pidForeground; /* pid of last process in foreground session */  
    UCHAR    fDynamicSched; /* dynamic variation flag */  
    UCHAR    csecMaxWait;   /* max wait in seconds */  
    USHORT   cmsecMinSlice; /* minimum timeslice (milliseconds) */  
    USHORT   cmsecMaxSlice; /* maximum timeslice (milliseconds) */  
    USHORT   bootdrive;     /* drive from which the system was booted */  
    UCHAR    amecRAS[32];   /* system trace major code flag bits */  
    UCHAR    csgWindowableVioMax; /* maximum number of VIO windowable sessions */  
    UCHAR    csgPMMMax;     /* maximum number of pres. services sessions */  
  
} GINFOSEG;
```

# DosGetInfoSeg —

## Get Address of System Variables Segment

```
typedef struct _LINFOSEG {
    PID    pidCurrent;      /* current process id */
    PID    pidParent;      /* process id of parent */
    USHORT prtyCurrent;     /* priority of current thread */
    TID    tidCurrent;     /* thread ID of current thread */
    USHORT sgCurrent;      /* session */
    UCHAR  rfProcStatus;   /* process status */
    UCHAR  dummy1;
    BOOL   fForeground;    /* current process has keyboard focus */
    UCHAR  typeProcess;    /* process type */
    UCHAR  dummy2;
    SEL    selEnvironment; /* environment selector */
    USHORT offCmdLine;     /* command line offset */
    USHORT cbDataSegment;  /* length of data segment */
    USHORT cbStack;        /* stack size */
    USHORT cbHeap;         /* heap size */
    HMODULE hmod;          /* module handle of the application */
    SEL    selDS;          /* data segment handle of the application */
} LINFOSEG;

#define INCL_DOSINFOSEG

USHORT rc = DosGetInfoSeg(GlobalSeg, LocalSeg);

PSEL    GlobalSeg;        /* Address to place global segment (selector) */
PSEL    LocalSeg;        /* Address to place local segment (selector) */

USHORT   rc;              /* return code */
```

## Assembler Language

```
GINFOSEG struc

    gis_time          dd ? ;time in seconds
    gis_msecs         dd ? ;milliseconds
    gis_hour          db ? ;hours
    gis_minutes       db ? ;minutes
    gis_seconds       db ? ;seconds
    gis_hundredths    db ? ;hundredths
    gis_timezone      dw ? ;minutes from UTC
    gis_cusecTimerInterval dw ? ;timer interval (units = 0.0001 seconds)
    gis_day           db ? ;day
    gis_month         db ? ;month
    gis_year          dw ? ;year
    gis_weekday       db ? ;day of week
    gis_uchMajorVersion db ? ;major version number
    gis_uchMinorVersion db ? ;minor version number
    gis_chRevisionLetter db ? ;revision letter
    gis_sgCurrent     db ? ;current foreground session
    gis_sgMax         db ? ;maximum number of sessions
    gis_cHugeShift     db ? ;shift count for huge elements
    gis_fProtectModeOnly db ? ;protect mode only indicator
    gis_pidForeground dw ? ;pid of last process in foreground session
    gis_fDynamicSched db ? ;dynamic variation flag
    gis_csecMaxWait    db ? ;max wait in seconds
    gis_cmsecMinSlice  dw ? ;minimum timeslice (milliseconds)
    gis_cmsecMaxSlice  dw ? ;maximum timeslice (milliseconds)
    gis_bootdrive      dw ? ;drive from which the system was booted
    gis_amecRAS        db 32 dup (?) ;system trace major code flag bits
    gis_csgWindowableVioMax db ? ;maximum number of VIO windowable sessions
    gis_csgPMMax       db ? ;maximum number of pres. services sessions

GINFOSEG ends
```

## DosGetInfoSeg — Get Address of System Variables Segment

LINFOSEG struc

```
lis_pidCurrent      dw ? ;current process id
lis_pidParent       dw ? ;process id of parent
lis_prtyCurrent     dw ? ;priority of current thread
lis_tidCurrent      dw ? ;thread ID of current thread
lis_sgCurrent       dw ? ;session
lis_rfProcStatus    db ? ;process status
lis_dummy1          db ? ;
lis_fForeground     dw ? ;current process has keyboard focus
lis_typeProcess     db ? ;process type
lis_dummy2          db ? ;
lis_selEnvironment  dw ? ;environment selector
lis_offCmdLine      dw ? ;command line offset
lis_cbDataSegment   dw ? ;length of data segment
lis_cbStack         dw ? ;stack size
lis_cbHeap          dw ? ;heap size
lis_hmod            dw ? ;module handle of the application
lis_selDS           dw ? ;data segment handle of the application
```

LINFOSEG ends

```
EXTRN DosGetInfoSeg:FAR
INCL_DOSINFOSEG EQU 1
```

```
PUSH@ WORD GlobalSeg ;Global segment selector (returned)
PUSH@ WORD LocalSeg  ;Local segment selector (returned)
CALL DosGetInfoSeg
```

Returns WORD

# DosGetMachineMode — Return Current Mode of Processor

FAPI

This call returns the current mode of the processor, whether the processor is running in the DOS mode or the OS/2 mode. This allows an application to determine whether a dynamic link call is valid or not.

DosGetMachineMode (MachineMode)

## Parameters

**MachineMode** (*PBYTE*) — output

Address of the value to indicate the current processor mode. This value may be:

Value	Definition
0	DOS mode
1	OS/2 mode.

**rc** (*USHORT*) — return

Return code description is:

0	NO_ERROR
---	----------

## Remarks

All dynamic link calls are available to an application if the MachineMode value indicates the program is in OS/2 mode. This method provides a self-tailoring application that allows the application to adapt to the execution environment by limiting or enhancing the functions it provides.

If the MachineMode value indicates the program is in DOS mode, the application is limited to a subset of dynamic link calls listed in the Family API.

## C Language

```
#define INCL_DOSQUEUES

USHORT rc = DosGetMachineMode(MachineMode);

PBYTE MachineMode; /* Processor mode (returned) */

USHORT rc; /* return code */
```

## Assembler Language

```
EXTRN DosGetMachineMode:FAR
INCL_DOSQUEUES EQU 1

PUSH@ BYTE MachineMode ;Processor mode (returned)
CALL DosGetMachineMode

Returns WORD
```

## DosGetMessage – Retrieve System Message with Variable Text

This call retrieves a message from a message file and inserts variable information into the body of the message.

**DosGetMessage** (*IvTable*, *IvCount*, *DataArea*, *DataLength*, *MsgNumber*, *FileName*,  
*MsgLength*)

### Parameters

**IvTable** (*PCHAR FAR \**) – input

Address of a list of double-word pointers. Each pointer points to an ASCIIZ or null-terminated DBCS string (variable insertion text). 0 to 9 strings can be present.

**IvCount** (*USHORT*) – input

Count of variable insertion text strings is 0–9. If *IvCount* is 0, *IvTable* is ignored.

**DataArea** (*PCHAR*) – output

Address of the requested message. If the message is too long to fit in the caller's buffer, as much of the message text is returned as possible, with the appropriate error return code.

**DataLength** (*USHORT*) – input

Length, in bytes, of the user's storage area.

**MsgNumber** (*USHORT*) – input

Requested message number.

**FileName** (*PSZ*) – input

Address of the optional drive, path, and filename of the file where the message can be found. If messages are bound to the .EXE file using MSGBIND utility, then filename is the name of the message file from which the messages are extracted.

**MsgLength** (*PUSHORT*) – output

Address of the length, in bytes, of the message.

**rc** (*USHORT*) – return

Return code descriptions are:

0	NO_ERROR
2	ERROR_FILE_NOT_FOUND
206	ERROR_FILENAME_EXCED_RANGE
316	ERROR_MR_MSG_TOO_LONG
317	ERROR_MR_MID_NOT_FOUND
318	ERROR_MR_UN_ACC_MSGF
319	ERROR_MR_INV_MSFG_FORMAT
320	ERROR_MR_INV_IVCOUNT
321	ERROR_MR_UN_PERFORM

### Remarks

If *IvCount* is greater than 9, *DosGetMessage* returns an error that indicates *IvCount* is out of range. If the numeric value of *x* in the %*x* sequence for %1through%9 is less than or equal to *IvCount*, text insertion by substitution for %*x*, is performed for all occurrences of %*x* in the message. Otherwise text insertion is ignored and the %*x* sequence is returned in the message unchanged. Text insertion is performed for all text strings defined by *IvCount* and *IvTable*.

Variable data insertion is not dependent on blank character delimiters nor are blanks automatically inserted.

For warning and error messages, the message ID (seven alphanumeric characters consisting of three upper case characters as the component ID, concatenated with a four-digit message number) followed by a colon and a blank character are returned to the caller as part of the message text. (*DosGetMessage*

# DosGetMessage —

## Retrieve System Message with Variable Text

determines the type of message based on the message classification generated in the output file of the MKMSGF utility.)

The following is an example of a sample error message returned with the message ID:

```
SYS0100: File not found
```

DosGetMessage retrieves messages previously prepared by the utility MKMSGF to create a message file, or MSGBIND to bind a message segment to an .EXE file. DosGetMessage tries to retrieve the message from RAM in the message segment bound to the .EXE program. If the message cannot be found, DosGetMessage retrieves the message from the message file on DASD (direct access storage device, such as a diskette or fixed-disk).

If the file name is not a fully-qualified name, DosGetMessage searches the following directories for default drive and path:

- The system root directory
- The current working directory
- Directories listed in the DPATH statement (OS/2 mode only)
- Directories listed in the APPEND statement (DOS mode only).

If a message cannot be retrieved because of a DASD error or file not found condition, a default message is placed in the user's buffer area. A message is placed in the buffer area based on the following error conditions:

- The message number cannot be found in the message file.
- The message file cannot be found.
- The system detected a disk error trying to access the message file, or the message file format is incorrect.
- IvCount is out of range.
- A system error occurred trying to allocate storage.

When these conditions occur, the default message allows the application program to issue a message and prompt that enables recovery opportunity.

The residency of the message in RAM (EXE bound) or on DASD is transparent to the caller and handled by DosGetMessage. In either case the message is referenced by message number and file name.

In order for DosGetMessage to be properly resolved, an application must be linked with DOSCALLS.LIB.

### Family API Considerations

Some options operate differently in the DOS mode than in OS/2 mode. Therefore, the following restriction applies to DosGetMessage when coding for the DOS mode:

If the message file name is not a fully qualified name, DosGetMessage searches the root directory of the default drive for the message file, instead of the root directory of the startup drive.

### C Language

```
#define INCL_DOSMISC
```

```
USHORT rc = DosGetMessage(IvTable, IvCount, DataArea, DataLength,  
                          MsgNumber, FileName, MsgLength);  
  
PCHAR FAR * IvTable; /* Table of variables to insert */  
USHORT IvCount; /* Number of variables */  
PCHAR DataArea; /* Message buffer (returned) */  
USHORT DataLength; /* Length of buffer */  
USHORT MsgNumber; /* Number of the message */  
PSZ FileName; /* Message file path name string */  
PUSHORT MsgLength; /* Length of message (returned) */  
  
USHORT rc; /* return code */
```

## **DosGetMessage — Retrieve System Message with Variable Text**

### **Assembler Language**

```
EXTRN DosGetMessage:FAR
INCL_DOSMISC EQU 1

PUSH@ OTHER IvTable      ;Table of variables to insert
PUSH WORD IvCount        ;Number of variables
PUSH@ OTHER DataArea     ;Message buffer (returned)
PUSH WORD DataLength     ;Length of buffer
PUSH WORD MsgNumber      ;Number of the message
PUSH@ ASCIIZ FileName    ;Message file path name string
PUSH WORD MsgLength      ;Length of message (returned)
CALL DosGetMessage

Returns WORD
```



# DosGetModHandle – Get Dynamic Link Module Handle

---

This call returns a handle to a previously loaded dynamic link module.

**DosGetModHandle (ModuleName, ModuleHandle)**

## Parameters

**ModuleName (PSZ)** – input

Address of the ASCIIZ name string containing the dynamic link module name or the pathname string.

The filename extension used for dynamic link libraries when a module name is provided is .DLL.

When a pathname is provided, the module name may have any extension.

**ModuleHandle (PHMODULE)** – output

Address of the handle for the dynamic link module.

**rc (USHORT)** – return

Return code descriptions are:

<b>0</b>	<b>NO_ERROR</b>
<b>95</b>	<b>ERROR_INTERRUPT</b>
<b>123</b>	<b>ERROR_INVALID_NAME</b>
<b>126</b>	<b>ERROR_MOD_NOT_FOUND</b>

## Remarks

If a module name is provided, it must match the name of a module residing in the LIBPATH that is currently loaded. Otherwise an error code is returned. If a pathname is provided, the expanded pathname must match the full pathname of a module that is currently loaded.

The handle returned by this call can be used with DosGetModName. It should not be used with DosGetProcAddress for access to the already loaded dynamic link module. Instead, DosLoadModule should be issued to ensure that the calling process is attached to the module.

## C Language

```
#define INCL_DOSMODULEMGR
```

```
USHORT rc = DosGetModHandle(ModuleName, ModuleHandle);
```

```
PSZ      ModuleName;    /* Module name string */
```

```
PHMODULE ModuleHandle;  /* Module handle (returned) */
```

```
USHORT   rc;            /* return code */
```

## Assembler Language

```
EXTRN DosGetModHandle:FAR
```

```
INCL_DOSMODULEMGR EQU 1
```

```
PUSH@ ASCIIZ ModuleName ;Module name string
```

```
PUSH@ WORD ModuleHandle ;Module handle (returned)
```

```
CALL DosGetModHandle
```

```
Returns WORD
```

# DosGetModName – Get Dynamic Link Module Name

This call returns the fully qualified drive, path, file name, and extension associated with a referenced module handle.

**DosGetModName (ModuleHandle, BufferLength, Buffer)**

## Parameters

**ModuleHandle** (*HMODULE*) – input

Handle of the dynamic link module being referenced.

**BufferLength** (*USHORT*) – input

Maximum length of the buffer, in bytes, where the name is stored.

**Buffer** (*PCHAR*) – output

Address of the buffer where the fully qualified drive, path, filename, and extension of the dynamic link module are stored.

**rc** (*USHORT*) – return

Return code descriptions are:

0	NO_ERROR
6	ERROR_INVALID_HANDLE
24	ERROR_BAD_LENGTH
95	ERROR_INTERRUPT

## Remarks

The handle returned by either a `DosGetModHandle` or a `DosLoadModule` request can be used with this call. An error is returned if the buffer is not large enough for the module name.

## C Language

```
#define INCL_DOSMODULEMGR

USHORT rc = DosGetModName(ModuleHandle, BufferLength, Buffer);

HMODULE      ModuleHandle; /* Module handle */
USHORT       BufferLength;  /* Buffer length */
PCHAR        Buffer;       /* Buffer (returned) */

USHORT       rc;           /* return code */
```

## Assembler Language

```
EXTRN DosGetModName:FAR
INCL_DOSMODULEMGR EQU 1

PUSH WORD ModuleHandle ;Module handle
PUSH WORD BufferLength ;Buffer length
PUSH@ OTHER Buffer ;Buffer (returned)
CALL DosGetModName

Returns WORD
```

---

This call returns the current process ID, thread ID, and the process ID of the parent process.

<b>DosGetPID (ProcessIDs)</b>
-------------------------------

## Parameters

**ProcessIDs** (*PPIDINFO*) — output

Address of the structure where the ID information is returned.

**pid** (*PID*)

Current process identifier.

**ttd** (*TID*)

Thread (of the current process) identifier.

**pidParent** (*PID*)

Parent process (of the current process) identifier.

**rc** (*USHORT*) — return

Return code description is:

0            NO\_ERROR

## Remarks

The process ID may be used to generate uniquely named temporary files, or for communication with signals. For more information on signals, see *DosFlagProcess* and *DosSendSignal*.

In the OS/2 environment, thread IDs are used with calls that manipulate threads in the current process. For more information, see *DosSuspendThread*, *DosResumeThread*, *DosGetPrty*, and *DosSetPrty*.

If the application is executing in the OS/2 environment, it is more efficient to obtain these variables by calling *DosGetInfoSeg* instead of *DosGetPID*. However, applications written to the family API cannot depend on the availability of *DosGetInfoSeg*.

To get an ID for a process other than the current process or its parent process, issue *DosGetPPID*.

## C Language

```
typedef struct _PIDINFO {    /* pidi */

    PID pid;                /* current process' process ID */
    TID tid;                /* current process' thread ID */
    PID pidParent;          /* process ID of the parent */

} PIDINFO;

#define INCL_DOSPROCESS

USHORT    rc = DosGetPID(ProcessIDsArea);

PPIDINFO    ProcessIDsArea; /* Process IDs (returned) */

USHORT    rc;                /* return code */
```

**Assembler Language**

```
PIDINFO struc
    pidi_pid      dw ? ;current process' process ID
    pidi_tid      dw ? ;current process' thread ID
    pidi_pidParent dw ? ;process ID of the parent
PIDINFO ends

EXTRN DosGetPID:FAR
INCL_DOSPROCESS    EQU 1

PUSH@ OTHER ProcessIDsArea ;Process IDs (returned)
CALL  DosGetPID

Returns NONE
```

**Example**

The following example demonstrates how to create a process, obtain process ID information, and kill a process. Process1 invokes process2 to run asynchronously. It obtains and prints some PID information, and then kills process2.

# DosGetPID — Return Process ID

FAPI

```
/* ---- process1.c ---- */

#define INCL_DOSPROCESS

#include <os2.h>

#define START_PROGRAM "process2.exe" /* Program pointer */

main()
{
    CHAR          ObjFail [50];      /* Object name buffer */
    RESULTCODES   ReturnCodes;       /*
    PIDINFO       PidInfo;           */
    PID           ParentID;          /*
    USHORT        rc;

    printf("Process1 now running. \n");

    /** Start a child process. **/
    if(!DosExecPgm(ObjFail,          /* Object name buffer */
                  sizeof(ObjFail),  /* Length of obj. name buffer */
                  EXEC_ASYNC,       /* Execution flag - asynchronous */
                  NULL,             /* No args. to pass to process2 */
                  NULL,             /* Process2 inherits process1's environment */
                  &ReturnCodes,     /* Ptr. to resultcodes struct. */
                  START_PROGRAM))   /* Name of program file */
        printf("Process2 started. \n");

    /** Obtain Process ID information and print it **/
    if(!rc=DosGetPID(&PidInfo))     /* Process ID's (returned) */
        printf("DosGetPID: current process ID is %d; thread ID is %d; parent process ID is %d.\n",
              PidInfo.pid, PidInfo.tid, PidInfo.pidParent);
    if(!rc=DosGetPPID(
        ReturnCodes.codeTerminate, /* Process whose parent is wanted */
        &ParentID))               /* Address to put parent's PID */
        printf("Child process ID is %d; Parent process ID is %d.\n",
              ReturnCodes.codeTerminate, ParentID);

    /** Terminate process2 **/
    if(!rc=DosKillProcess(DKP_PROCESSTREE, /* Action code - kill process and descendants */
                          ReturnCodes.codeTerminate)) /* PID of root of process tree */
        printf("Process2 terminated by process1.\n");
}

/* ---- process2.c ---- */

#define INCL_DOSPROCESS

#include <os2.h>

#define SLEEPTIME 500L
#define RETURN_CODE 0

main()
{
    printf("Process2 now running.\n");

    /** Sleep to allow process1 to kill it */
    DosSleep(SLEEPTIME);          /* Sleep interval */
    DosExit(EXIT_PROCESS,         /* Action Code */
            RETURN_CODE);         /* Result Code */
}
```

## DosGetPPID – Get a Process's Parent's PID

---

This call allows the caller to obtain the parent process ID for any process.

<b>DosGetPPID (PID, PPID)</b>
-------------------------------

### Parameters

**PID** (*USHORT*) – input

The process ID of the process to find the parent PID.

**PPID** (*PUSHORT*) – output

Address of a word where the process ID of the parent of the indicated process is placed.

**rc** (*USHORT*) – return

Return code descriptions are:

0	NO_ERROR
303	ERROR_INVALID_PROCID

### C Language

```
#define INCL_DOSPROCESS
```

```
USHORT rc = DosGetPPID(PID, PPID);
```

```
USHORT      PID;          /* Process whose parent is wanted */
PUSHORT      PPID;         /* Address to put parent's PID */
```

```
USHORT      rc;           /* return code */
```

### Assembler Language

```
EXTRN DosGetPPID:FAR
INCL_DOSPROCESS EQU 1
```

```
PUSH WORD PID      ;Process whose parent is wanted
PUSH@ WORD PPID     ;Address to put parent's PID
CALL DosGetPPID
```

Returns NONE

### Example

The following example demonstrates how to create a process, obtain process ID information, and kill a process. Process1 invokes process2 to run asynchronously. It obtains and prints some PID information, and then kills process2.

## DosGetPPID — Get a Process's Parent's PID

```

/* ---- process1.c ---- */

#define INCL_DOSPROCESS

#include <os2.h>

#define START_PROGRAM "process2.exe" /* Program pointer */

main()
{
    CHAR          ObjFail [50];      /* Object name buffer */
    RESULTCODES   ReturnCodes;      /*
    PIDINFO       PidInfo;           /*
    PID           ParentID;          /*
    USHORT        rc;

    printf("Process1 now running. \n");

    /** Start a child process. **/
    if(!DosExecPgm(ObjFail,          /* Object name buffer */
                  sizeof(ObjFail),  /* Length of obj. name buffer */
                  EXEC_ASYNC,       /* Execution flag - asynchronous */
                  NULL,             /* No args. to pass to process2*/
                  NULL,             /* Process2 inherits process1's environment */
                  &ReturnCodes,     /* Ptr. to resultcodes struct. */
                  START_PROGRAM)))  /* Name of program file */
        printf("Process2 started. \n");

    /** Obtain Process ID information and print it **/
    if(!rc=DosGetPID(&PidInfo))     /* Process ID's (returned) */
        printf("DosGetPID: current process ID is %d; thread ID is %d; parent process ID is %d.\n",
              PidInfo.pid, PidInfo.tid, PidInfo.pidParent);
    if(!rc=DosGetPPID(
        ReturnCodes.codeTerminate, /* Process whose parent is wanted */
        &ParentID)))              /* Address to put parent's PID */
        printf("Child process ID is %d; Parent process ID is %d.\n",
              ReturnCodes.codeTerminate, ParentID);

    /** Terminate process2 **/
    if(!rc=DosKillProcess(DKP_PROCESSTREE, /* Action code - kill process and descendants */
                          ReturnCodes.codeTerminate)) /* PID of root of process tree */
        printf("Process2 terminated by process1.\n");
}

/* ---- process2.c ---- */

#define INCL_DOSPROCESS

#include <os2.h>

#define SLEEPTIME 500L
#define RETURN_CODE 0

main()
{
    printf("Process2 now running.\n");

    /** Sleep to allow process1 to kill it */
    DosSleep(SLEEPTIME);          /* Sleep interval */
    DosExit(EXIT_PROCESS,         /* Action Code */
            RETURN_CODE);         /* Result Code */
}

```

# DosGetProcAddress – Get Dynamic Link Procedure Address

This call returns a far address to a desired procedure within a dynamic link module.

**DosGetProcAddress (ModuleHandle, ProcName, ProcAddress)**

## Parameters

**ModuleHandle** (*HMODULE*) – input

Handle of the dynamic link module.

**ProcName** (*PSZ*) – input

Address of a name string that contains the referenced procedure name.

Alternatively, if the selector portion of the pointer is null, the offset portion of the pointer is an explicit entry number (ordinal) within the dynamic link module.

DosGetProcAddress for entries within the DOSCALLS module are only supported for ordinal references.

References to the DOSCALLS module by name strings are not supported and return an error.

Dynamic link ordinal numbers for DOSCALLS routines are resolved by linking with DOSCALLS.LIB.

**ProcAddress** (*PFN FAR \**) – output

Procedure address.

**rc** (*USHORT*) – return

Return code descriptions are:

0	NO_ERROR
6	ERROR_INVALID_HANDLE
95	ERROR_INTERRUPT
127	ERROR_PROC_NOT_FOUND

## Remarks

A 32-bit address, consisting of a selector and offset, is returned for a specified procedure.

To free the dynamic link module, issue DosFreeModule. After DosFreeModule is issued, procedure entry addresses returned for this handle are no longer valid.

Other run-time dynamic link calls are DosLoadModule, DosGetModName, and DosGetModHandle.

## C Language

```
#define INCL_DOSMODULEMGR
```

```
USHORT rc = DosGetProcAddress(ModuleHandle, ProcName, ProcAddress);
```

```
HMODULE      ModuleHandle; /* Module handle */
PSZ           ProcName;    /* Module name string */
PFN FAR *     ProcAddress; /* Procedure address (returned) */

USHORT        rc;          /* return code */
```

## Assembler Language

```
EXTRN DosGetProcAddress:FAR
INCL_DOSMODULEMGR EQU 1

PUSH WORD ModuleHandle ;Module handle
PUSH@ ASCIIZ ProcName ;Module name string
PUSH@ DWORD ProcAddress ;Procedure address (returned)
CALL DosGetProcAddress
```

Returns WORD



# DosGetPrtly – Get Process's Priority

This call gets the priority of the initial thread of execution for any process, or any thread in the current process.

DosGetPrtly (Scope, Priority, ID)

## Parameters

**Scope** (*USHORT*) – input

Scope of priority request. Used to define the scope of the request.

Value	Definition
0	Thread 1 (the initial thread of execution) of the indicated process.
2	Any thread of the current process.

**Priority** (*PUSHORT*) – output

Address of the base priority of the thread identified by ID. The high-order byte of this word is set equal to the priority class. The low-order byte is set equal to the priority level.

**ID** (*USHORT*) – input

A process ID (scope = 0) or a thread ID (scope = 2). If this operand is equal to 0, the current process or thread is assumed.

**rc** (*USHORT*) – return

Return code descriptions are:

0	NO_ERROR
303	ERROR_INVALID_PROCID
308	ERROR_INVALID_SCOPE
309	ERROR_INVALID_THREADID

## Remarks

If scope = 0 (process) the priority of the first thread of a process is returned. If that thread has terminated, the ERROR\_INVALID\_THREAD\_ID error code is returned.

A thread's priority is changed by issuing DosSetPrtly. A process can change the priority of all the threads of any process whose process ID is known to the thread. It can also change the priority of another thread in its process, or all the threads of the current or a child process, as well as all its descendants.

DosGetPID and DosGetPPID are useful for obtaining process and thread IDs. DosGetPID gets the ID of the current thread, the process ID of its current process, or the process ID of the parent process of the current process. DosGetPPID gets the parent process ID of any specified process.

## C Language

```
#define INCL_DOSPROCESS
```

```
USHORT rc = DosGetPrtly(Scope, Priority, ID);
```

```
USHORT    Scope;    /* Indicate scope of query */
PUSHORT   Priority;  /* Address to put priority (returned) */
USHORT    ID;       /* Process or thread ID */

USHORT    rc;        /* return code */
```

# DosGetPrty – Get Process's Priority

## Assembler Language

```
EXTRN DosGetPrty:FAR
INCL_DOSPROCESS EQU 1

PUSH WORD Scope ;Indicate scope of query
PUSH@ WORD Priority ;Priority (returned)
PUSH WORD ID ;Process or thread ID
CALL DosGetPrty
```

Returns WORD

## Example

The following example illustrates how to obtain the priority of a thread and how to change the priority. The main thread creates Thread2 and allows it to begin executing. Thread2 iterates through a loop that prints a line and then sleeps, relinquishing its time slice to the main thread. After one or two iterations by Thread2, the main thread obtains Thread2's priority information and prints it. It then raises Thread2's priority to fixed-high, and increments the level by ten. Since Thread2 is now at a high priority, it immediately finishes its remaining iterations before relinquishing control on a long sleep; at this point, the main thread re-examines Thread2's priority and reports its new priority level. In this example, it is helpful to understand how the DosSleep calls are used either to relinquish control of the processor, or to keep a thread alive (see DosTimerAsync or DosTimerStart for alternatives to DosSleep).

```
#define INCL_DOSPROCESS

#include <os2.h>

#define PRTYC_FIXEDHIGH 4 /* Priority class: fixed-high */
#define PRTY_DELTA 10 /* Priority delta: increase by 10 */
#define SEGSIZE 4000 /* Number of bytes requested in segment */
#define ALLOCFLAGS 0 /* Segment allocation flags - no sharing */
#define SLEEPSHORT 0L /* Sleep interval - 5 milliseconds */
#define SLEEPLONG 20L /* Sleep interval - 75 milliseconds */
#define RETURN_CODE 0 /* Return code for DosExit() */

VOID APIENTRY Thread2()
{
    USHORT i;

    /* Loop with four iterations */
    for(i=1; i<5; i++)
    {
        printf("In Thread2, i is now %d\n", i);

        /** Sleep to relinquish time slice to main thread **/
        DosSleep(SLEEPSHORT); /* Sleep interval */
    }
    DosExit(EXIT_THREAD, /* Action code - end a thread */
        RETURN_CODE); /* Return code */
}
```

## DosGetPrty — Get Process's Priority

```
main()
{
    USHORT    Priority;          /* Thread priority */
    USHORT    Class;            /* Priority class */
    USHORT    Level;            /* Priority level */
    SEL        ThreadStackSel;   /* Segment selector for thread stack */
    PBYTE     StackEnd;          /* Ptr. to end of thread stack */
    USHORT    rc;

    /* Allocate segment for thread stack; this is better than just */
    /* declaring an array of bytes to use as a stack. Make pointer eos. */

    rc = DosAllocSeg(SEGSIZE,          /* Number of bytes requested */
                    &ThreadStackSel, /* Segment selector (returned) */
                    ALLOCFLAGS);      /* Allocation flags */
    StackEnd = MAKEP(ThreadStackSel, SEGSIZE-1);

    /* Start Thread2 */
    if(! (DosCreateThread((PFNTHREAD) Thread2, /* Thread address */
                        &ThreadID, /* Thread ID (returned) */
                        StackEnd))) /* End of thread stack */
        printf("Thread2 created.\n");

    /** Sleep to allow Thread2 to execute **/
    if(! (DosSleep(SLEEPLONG))) /* Sleep interval */
        printf("Slept a little to let Thread2 execute.\n");

    /** Obtain Thread2's priority information and report it **/
    if(! (rc=DosGetPrty(PRTYS_THREAD, /* Scope - single thread */
                        &Priority, /* Address to put priority */
                        ThreadID))) /* ID - thread ID */
    {
        /* Extract priority class and level information */
        Class = HIBYTE(Priority);
        Level = LOBYTE(Priority);
        printf("Thread2: ID is %d, Priority Class is %d and Level is %d\n",
            ThreadID, Class, Level);
    }

    /** Raise Thread2's priority **/
    if(! (rc=DosSetPrty(PRTYS_THREAD, /* Scope - single thread */
                        PRTYC_FIXEDHIGH, /* Prty class - fixed-high */
                        PRTY_DELTA, /* Prty delta - increase by 10 */
                        ThreadID))) /* ID - thread ID */
    {
        /* Obtain Thread2' new priority information and report it */
        rc=DosGetPrty(PRTYS_THREAD, /* Scope - single thread */
                    &Priority, /* Address to put priority */
                    ThreadID); /* ID - thread ID */

        /* Extract priority class and level information */
        Class = HIBYTE(Priority);
        Level = LOBYTE(Priority);
        printf("Thread2: ID is %d, New Priority Class is %d and Level is %d\n",
            ThreadID, Class, Level);
    }
}
```

# DosGetResource – Get Resource Segment Selector

---

This call returns the segment selector of the specified resource segment.

<b>DosGetResource</b> ( <i>ModHandle</i> , <i>TypeID</i> , <i>NameID</i> , <i>Selector</i> )
--

## Parameters

**ModHandle** (*HMODULE*) – input

The location of the resource segment.

Value	Definition
-------	------------

0	The executable file of the current process.
---	---

≠0	A handle to a dynamic link module returned by <code>DosLoadModule</code> .
----	--

**TypeID** (*USHORT*) – input

A 16 bit resource type ID (see Remarks).

**NameID** (*USHORT*) – input

A 16 bit resource name ID (see Remarks).

**Selector** (*PSEL*) – output

The address of a word where the resource segment selector is returned.

**rc** (*USHORT*) – return

Return code descriptions are:

0	NO_ERROR
---	----------

6	ERROR_INVALID_HANDLE
---	----------------------

## Remarks

Resource segments are read-only data segments that can be accessed dynamically at run time. The access key consists of two 16-bit numbers, the first of which is a type ID and the second, a name ID. These numbers are similar in concept to the file extension and file name portions of a file name.

The advantage of resource segments is that they can be bundled into an application's executable file, so a single file contains all of the code and data for an application.

It is recommended that resource segments obtained through `DosGetResource` only be freed using `DosFreeSeg`.

OS/2 Version 1.2 added two new functions, `DosGetResource2` and `DosFreeResource`, to load and free an application specific resource. `DosGetResource2` returns a far pointer to a resource rather than the selector returned via `DosGetResource`. It is recommended that applications targeted for OS/2 1.2 use `DosGetResource2` and `DosFreeResource`. Applications that wish to execute on OS/2 1.1 and 1.2 should use the OS/2 run-time dynamic link capabilities, `DosLoadModule` and `DosGetProcAddr`, to get the address of `DosGetResource2` and `DosFreeResource` when executing on OS/2 1.2. If the `DosGetProcAddr` call to obtain the address of `DosGetResource2` and `DosFreeResource` fails, the application can call `DosGetResource` and `DosFreeSeg`. Applications that use `DosGetResource2` and `DosFreeResource` allow OS/2 to optimize memory allocation associated with the applications resource.

# DosGetResource — Get Resource Segment Selector

## C Language

```
#define INCL_DOSRESOURCES

USHORT rc = DosGetResource(ModHandle, TypeID, NameID, Selector);

HMODULE      ModHandle;    /* Module handle to get resource from */
USHORT       TypeID;       /* 16 bit resource type ID */
USHORT       NameID;       /* 16 bit resource name ID */
PSEL         Selector;     /* where to return selector */

USHORT       rc;           /* return code */
```

## Assembler Language

```
EXTRN DosGetResource:FAR
INCL_DOSRESOURCES EQU 1

PUSH WORD ModHandle ;Module handle to get resource from
PUSH WORD TypeID ;16 bit resource type ID
PUSH WORD NameID ;16 bit resource name ID
PUSH@ WORD Selector ;Resource selector (returned)
CALL DosGetResource
```

Returns WORD

## DosGetResource2 – Get Resource Address

This call returns the far address of the specified resource.

**DosGetResource2 (ModHandle, TypeID, NameID, ResAddr)**

### Parameters

**ModHandle** (*HMODULE*) – input  
The location of the resource.

Value	Definition
0	The executable file of the current process.
≠0	A handle to a dynamic link module returned by DosLoadModule.

**TypeID** (*USHORT*) – input  
A 16 bit resource type ID (see Remarks).

**NameID** (*USHORT*) – input  
A 16 bit resource name ID (see Remarks).

**ResAddr** (*PULONG*) – output  
Address of the resource address.

**rc** (*USHORT*) – return  
Return code descriptions are:

0	NO_ERROR
6	ERROR_INVALID_HANDLE

### Remarks

A resource is read-only data generated by the Resource Compiler that can be accessed dynamically at run time. The access key consists of two 16-bit numbers, the first of which is a type ID and the second, a name ID. These numbers are similar in concept to the file extension and file name portions of a file name.

The advantage of a resource is that it can be bundled into an application's executable file, so a single file contains all of the code and data for an application.

Resource segments obtained through DosGetResource2 should only be freed using DosFreeResource.

### C Language

```
#define INCL_DOSRESOURCES2
```

```
USHORT rc = DosGetResource2(ModHandle, TypeID, NameID, ResAddr);
```

```
HMODULE    ModHandle;    /* Module handle to get resource from */
USHORT      TypeID;       /* 16 bit resource type ID */
USHORT      NameID;       /* 16 bit resource name ID */
PULONG      ResAddr;      /* where to return resource address */

USHORT      rc;           /* return code */
```

# DosGetResource2 – Get Resource Address

## Assembler Language

```
EXTRN  DosGetResource2:FAR
INCL_DOSRESOURCES2  EQU 1

PUSH  WORD    ModHandle    ;Module handle to get resource from
PUSH  WORD    TypeID       ;16 bit resource type ID
PUSH  WORD    NameID       ;16 bit resource name ID
PUSH@  DWORD   ResAddr     ;Resource address (returned)
CALL  DosGetResource2
```

Returns WORD

# DosGetSeg — Access Segment

This call accesses shared memory allocated by a DosAllocSeg or DosAllocHuge call.

**DosGetSeg (Selector)**

## Parameters

**Selector (SEL)** — input

Parameter used to get access to a segment.

**rc (USHORT)** — return

Return code descriptions are:

0	NO_ERROR
5	ERROR_ACCESS_DENIED

## Remarks

Before a process can call DosGetSeg to access shared memory allocated by a DosAllocSeg or DosAllocHuge request, it must be given a selector by the process that allocated the memory. The giving process obtains this selector by calling DosGiveSeg. It then passes the selector returned by DosGiveSeg to the recipient process, using some form of interprocess communication. If the recipient has created a queue with DosCreateQueue, the giving process can write the selector to the queue and then free the segment with a call to DosFreeSeg. The recipient removes the selector from the queue with a DosReadQueue and calls DosGetSeg, specifying the passed selector.

If at the time the shared segment is allocated, it is also specified as discardable, it is automatically locked for access by the caller. The caller may free the segment for discard by a DosUnlockSeg call. A process that gains access to the discardable shared segment by calling DosGetSeg has to lock the segment with a DosLockSeg request. However, DosLockSeg may return an error, indicating the segment is already locked. In this case, the process calls DosUnlockSeg repetitively, until the segment is fully unlocked. The process then locks the segment for its own use. Locking is an attribute of the segment, not the processes using the segment.

To access named shared memory allocated with a DosAllocShrSeg request, a process issues DosGetShrSeg.

## C Language

```
#define INCL_DOSMEMMGR

USHORT rc = DosGetSeg(Selector);

SEL      Selector;      /*Selector to access */

USHORT   rc;             /* return code */
```

## Assembler Language

```
EXTRN DosGetSeg:FAR
INCL_DOSMEMMGR EQU 1

PUSH WORD Selector ;Selector to access
CALL DosGetSeg

Returns WORD
```



# DosGetShrSeg — Access Shared Segment

---

This call accesses a shared memory segment previously allocated by another process.

**DosGetShrSeg (Name, Selector)**

## Parameters

**Name (PSZ)** — input

Address of the name string associated with the shared memory segment to be accessed. The name is an ASCII string in the format of an OS/2 filename in a subdirectory called \SHAREMEM\, for example, \SHAREMEM\PUBLIC.DAT.

**Selector (PSEL)** — output

Address of the selector for the shared memory segment.

**rc (USHORT)** — return

Return code descriptions are:

0	NO_ERROR
2	ERROR_FILE_NOT_FOUND
4	ERROR_TOO_MANY_OPEN_FILES
123	ERROR_INVALID_NAME

## Remarks

DosGetShrSeg provides access to a named shared segment allocated by another process with DosAllocShrSeg. The selector returned by DosGetShrSeg is the same as the one returned by the DosAllocShrSeg call.

A usage count is maintained for a named shared segment. Issuing DosGetShrSeg increments the count, and issuing DosFreeSeg decrements the count. When the usage count equals zero, the named shared segment is deallocated. Once the segment has been deallocated, it must be reinitialized by a call to DosAllocShrSeg.

To access shared memory that is allocated by another process with DosAllocSeg and DosAllocHuge requests, a process issues DosGetSeg.

## C Language

```
#define INCL_DOSMEMMGR

USHORT rc = DosGetShrSeg(Name, Selector);

PSZ      Name;      /* Name string */
PSEL     Selector;  /* Selector of shared segment */

USHORT   rc;        /* return code */
```

## Assembler Language

```
EXTRN DosGetShrSeg:FAR
INCL_DOSMEMMGR EQU 1

PUSH@ ASCIIZ Name      ;Name string
PUSH@ WORD Selector    ;Selector of shared segment (returned)
CALL DosGetShrSeg

Returns WORD
```

---

This call returns the OS/2 version number.

<b>DosGetVersion (VersionWord)</b>
------------------------------------

## Parameters

**VersionWord** (*PUSHORT*) – output

Address of the OS/2 version number. The version is stored as two bytes, with the minor version in the first byte and major version in the second byte.

**rc** (*USHORT*) – return

Return code description is:

0            NO\_ERROR

## C Language

```
#define INCL_DOSMISC
```

```
USHORT rc = DosGetVersion(VersionWord);
```

```
PUSHORT        VersionWord;    /* Version number (returned) */
```

```
USHORT        rc;               /* return code */
```

## Assembler Language

```
EXTRN DosGetVersion:FAR  
INCL_DOSMISC        EQU 1
```

```
PUSH@ WORD    VersionWord    ;Version number(returned)  
CALL    DosGetVersion
```

Returns WORD

# DosGetVersion —

## Get OS/2 Version Number

FAPI

### Example

The following example shows how one may obtain information for program initialization. The program locates the environment segment and prints the name of the command from the command line. It then obtains the OS/2 version number and prints it.

```
#define INCL_DOS

#include <os2.h>

#define ENVVARNAME "PATH"

main()
{
    SEL      EnvSel;           /* Environment segment selector (returned) */
    USHORT   CmdOffset;        /* Offset into env. seg. of command line (returned) */
    PSZ FAR  *Commandline;     /* Pointer made by EnvSel and CmdOffset */
    USHORT   Version;          /* Version numbers (returned) */
    BYTE     MajorVer;         /* Major version number */
    BYTE     MinorVer;         /* Minor version number */
    USHORT   rc;               /* return code */

    /** Locate environment segment and offset of command line. **/

    if(! (rc=DosGetEnv(&EnvSel, /* Env. seg. selector (returned) */
                      &CmdOffset))) /* Offset of command line (returned) */
        printf("Environment located; selector is %x offset is %x\n", EnvSel,
              CmdOffset);

    /** Use a macro to make a far pointer out of selector:offset pair. **/
    /** Notice the far-string pointer specification (%Fs) used to print **/

    Commandline = MAKEP(EnvSel, CmdOffset);
    printf("Command entered is %Fs.\n", Commandline);

    /** Obtain and print version info; use macros to extract info. **/
    /** We need to divide by 10 to obtain true version numbers.   **/

    if(! (rc=DosGetVersion(&Version)))
    {
        MajorVer = HIBYTE(Version) / 10;
        MinorVer = LOBYTE(Version) / 10;
        printf("This is OS/2 version %d.%d\n", MajorVer, MinorVer);
    }
}
```

# DosGiveSeg – Give Access to Segment

This call allows another process to access shared memory allocated by a DosAllocSeg or DosAllocHuge call.

**DosGiveSeg (CallerSegSelector, ProcessID, RecipientSegSelector)**

## Parameters

**CallerSegSelector (SEL)** – input

Segment selector of the shared memory segment.

**ProcessID (PID)** – input

Process ID of the process to receive access to the shared memory segment.

**RecipientSegSelector (PSEL)** – output

Address of the recipient's segment selector.

**rc (USHORT)** – return

Return code descriptions are:

0	NO_ERROR
5	ERROR_ACCESS_DENIED
8	ERROR_NOT_ENOUGH_MEMORY

## Remarks

DosGiveSeg returns a selector that can be given to another process to access shared memory the giver has allocated by a DosAllocSeg or DosAllocHuge call. The giving process passes the recipient's selector to the intended sharer by some means of interprocess communication. If the recipient has created a queue with DosCreateQueue, the giver can issue DosWriteQueue, specifying the queue name, and pass the selector in this manner.

If the memory being given was allocated by a DosAllocHuge request, the CallerSegSelector must be the base selector returned by DosAllocHuge. When the caller passes the selector returned in RecipientSegSelector to the intended sharer, this selector has addressability only to the first segment in the sharer's address space of the huge allocation. However, the recipient can call DosGetHugeShift and use the shift count returned to calculate the selectors for the remaining segments.

## C Language

```
#define INCL_DOSMEMMGR
```

```
USHORT rc = DosGiveSeg(CallerSegSelector, ProcessID, RecipientSegSelector);
```

```
SEL      CallerSegSelector;    /* Caller's segment selector */
PID      ProcessID;            /* Process ID of recipient */
PSEL     RecipientSegSelector; /* Recipient's segment selector (returned) */
```

```
USHORT   rc;                  /* return code */
```

## Assembler Language

```
EXTRN DosGiveSeg:FAR
```

```
INCL_DOSMEMMGR EQU 1
```

```
PUSH WORD CallerSegSelector ;Caller's segment selector
PUSH WORD ProcessID         ;Process ID of recipient
PUSH@ WORD RecipientSegSelector ;Recipient's segment selector (returned)
CALL DosGiveSeg
```

Returns WORD

# DosHoldSignal — Disable/Enable Signals

FAPI

This call temporarily disables or enables signal processing for the current process.

**DosHoldSignal (ActionCode)**

## Parameters

**ActionCode** (*USHORT*) — Input

Disables or enables signals intended for the current process.

Value	Definition
0	Signals are enabled
1	Signals are disabled.

**rc** (*USHORT*) — return

Return code descriptions are:

0	NO_ERROR
1	ERROR_INVALID_FUNCTION

## Remarks

DosHoldSignal with ActionCode = 1 causes signal processing (except central processing errors and numeric processor errors) to be postponed until a DosHoldSignal with ActionCode = 0 is issued. Any signals that occur while processing is disabled are recognized, but not accepted until signal recognition is enabled.

To allow for nesting of requests, a count of the number of outstanding DosHoldSignal requests with ActionCode = 1 are maintained.

DosHoldSignal is used by library routines, subsystems, and similar code that lock critical sections or temporarily reserve resources needed to prevent a signal from terminating a task. A process can have only one signal handling address for each signal. Dynalink routines should not have a signal handler (which might override a handler established by a calling process).

Signals can be held for a short period and should be released and re-held, if necessary. Their guidelines for proper use are similar to hardware interrupt counterparts such as the CLI/STI instructions.

## Family API Considerations

Some options operate differently in the DOS mode than in the OS/2 mode. Therefore, the following restriction applies to DosHoldSignal when coding for the DOS mode:

The only signal recognized in the DOS mode is SIGINTR (Ctrl-C) and SIGBREAK. Only SIGINTR and SIGBREAK are turned off by this call.

## C Language

```
#define INCL_DOSSIGNALS

USHORT rc = DosHoldSignal(ActionCode);

USHORT      ActionCode;    /* Indicate to Disable/Enable Signals */

USHORT      rc;            /* return code */
```

## Assembler Language

```

EXTRN DosHoldSignal:FAR
INCL_DOSSIGNALS EQU 1

PUSH WORD ActionCode ;Indicate to Disable/Enable Signals
CALL DosHoldSignal

```

Returns NONE

## Example

The following example illustrates the use of the Ctrl-C (SIGINTR) signal to signal time-critical events. Process1 invokes process2, which establishes a signal handler named CtrlC\_Handler() and waits, by blocking on a reserved RAM semaphore, for a signal from process1. A portion of process2 is immune to signalling.

```

#define INCL_DOSPROCESS
#define INCL_DOSSIGNALS

#include <os2.h>

#define SLEEPTIME 200L /* Sleep interval */
#define START_PROGRAM "process2.exe" /* Program name */

main()
{
    CHAR ObjFail[50];
    PSZ Args;
    PSZ Envs;
    RESULTCODES ReturnCodes;
    USHORT rc;

    /* Start process2 and check its PID */
    if(!DosExecPgm(ObjFail, /* Object name buffer */
                  sizeof(ObjFail), /* Length of obj. name buffer */
                  EXEC_ASYNC, /* Execution flag */
                  Args, /* Ptr. to argument string */
                  Envs, /* Ptr. to environment string */
                  &ReturnCodes, /* Ptr. to resultcodes struct. */
                  START_PROGRAM))) /* Name of program file */
    {
        printf("Process2 started.\n");
        printf("Process2 ID is %d\n", ReturnCodes.codeTerminate);

        /* Sleep to give time slice to process2 */
        DosSleep(SLEEPTIME); /* Sleep interval */

        /*** After process2 sets signal handler, send process2 a signal ***/
        if(!rc = DosSendSignal(ReturnCodes.codeTerminate, /* PID of process2 */
                              SIG_CTRLC)) /* Signal to send */
        {
            printf("Ctrl-C signal sent from Process1 to Process2.\n");
        }
    }
}

```

# DosHoldSignal — Disable/Enable Signals

FAP1

```
/* ----- process2.c ----- */

#define INCL_DOSPROCESS
#define INCL_DOSSIGNALS
#define INCL_DOSERRORS

#include <os2.h>

#define SLEEPTIME      50L
#define TIMEOUT        5000L

VOID APIENTRY CtrlC_Handler(arg1, arg2)    /** Define signal handler **/
{
    USHORT      arg1;
    USHORT      arg2;
    {
        printf("Handler for Ctrl-C now running.\n");
        return;
    }
}

main()
{
    ULONG      RamSem = 0L;    /** Allocate and initialize Ram Semaphore */
    ULONG far  *RamSemHandle = &RamSem;    /** Ram Semaphore handle */
    USHORT      rc;

    /** Establish signal handler */
    if(!rc=DosSetSigHandler((PFNSIGHANDLER) CtrlC_Handler,
        NULL,                /** Previous handler - ignored */
        NULL,                /** Previous action - ignored */
        SIGA_ACCEPT,         /** Request type */
        SIG_CTRLC))          /** Signal number */
        printf("Process2 has set Ctrl-C handler.\n");
    else
    {
        /** Error processing on rc */;
        /** Get semaphore for first time */
        if(!rc=DosSemRequest(RamSemHandle,    /** Semaphore handle */
            TIMEOUT))                        /** Timeout interval */
            printf("Semaphore obtained.\n");

        /**** Disable and then enable signal-handling ***/
        if(!rc=DosHoldSignal(HLDSIG_DISABLE)) /** Action code - disable */
        {
            printf("Signalling DISABLED.\n");

            /** Do signal-proof work here */
            if(!rc=DosHoldSignal(HLDSIG_ENABLE)) /** Action code - enable */
                printf("Signalling ENABLED.\n");
        }
        /** At this point, process1 may have sent a Ctrl-C signal. */
        /** Try to obtain semaphore again -- resulting in Timeout. */
        /** The Timeout, however, may be interrupted by the signal. */

        printf("Process2 will now wait on a Ramsem for a while.\n");
        if((rc=DosSemRequest(RamSemHandle,    /** Semaphore handle */
            TIMEOUT))                        /** Timeout interval */
            == ERROR_INTERRUPT)
            printf("Process2 interrupted while waiting, rc is %d.\n", rc);
    }
}
```

# DosInsMessage — Insert Variable Text Strings In Message

This call inserts variable text string information into the body of a message. This is useful when messages are loaded before insertion text strings are known.

**DosInsMessage** (*lvTable*, *lvCount*, *MsgInput*, *MsgInLength*, *DataArea*, *DataLength*, *MsgLength*)

## Parameters

**lvTable** (*PCHAR FAR \**) — input

List of double-word pointers. Each pointer points to an ASCIIZ or null terminated DBCS string (variable insertion text). 0 to 9 strings can be present.

**lvCount** (*USHORT*) — input

0–9 is the count of variable insertion text strings. If lvCount is 0, lvTable is ignored.

**MsgInput** (*PSZ*) — input

Address of the input message.

**MsgInLength** (*USHORT*) — input

Length, in bytes, of the input message.

**DataArea** (*PCHAR*) — output

Address of the user storage that returns the updated message. If the message is too long to fit in the caller's buffer, as much of the message text as possible is returned with the appropriate return code.

**DataLength** (*USHORT*) — input

Length, in bytes, of the user's storage area.

**MsgLength** (*PUSHORT*) — output

Address of the length, in bytes, of the updated message.

**rc** (*USHORT*) — return

Return code descriptions are:

<b>0</b>	<b>NO_ERROR</b>
<b>316</b>	<b>ERROR_MR_MSG_TOO_LONG</b>
<b>320</b>	<b>ERROR_MR_INV_IVCOUNT</b>

## Remarks

DosInsMessage returns an error indicating that lvCount is out of range when lvCount is greater than 9. A default message is also placed in the caller's buffer. Refer to "DosGetMessage — Retrieve System Message with Variable Text" on page 2-143 for details on the default messages. If the numeric value of x in the %x sequence for %1through%9 is less than or equal to lvCount, then text insertion, by substitution for %x, is performed for all occurrences of %x in the body of the message. Otherwise text insertion is ignored and the %x sequence is returned unchanged in the message. Text insertion is performed for all text strings defined by lvCount and lvTable.

Variable data insertion does not depend on a blank character delimiter nor are blanks automatically inserted.



# DosInsMessage — Insert Variable Text Strings In Message

FAPI

## C Language

```
#define INCL_DOSMISC

USHORT rc = DosInsMessage(IvTable, IvCount, MsgInput, MsgInLength,
                          DataArea, DataLength, MsgLength);

PCHAR FAR *    IvTable;    /* Table of variables to insert */
USHORT         IvCount;    /* Number of variables */
PSZ           MsgInput;    /* Address of input message */
USHORT        MsgInLength; /* Length of input message */
PCHAR         DataArea;    /* Updated message (returned) */
USHORT        DataLength;  /* Length of updated message buffer */
PUSHORT       MsgLength;   /* Length of updated message (returned) */

USHORT rc;                /* return code */
```

## Assembler Language

```
EXTRN DosInsMessage:FAR
INCL_DOSMISC EQU 1

PUSH@ OTHER IvTable    ;Table of variables to insert
PUSH WORD IvCount      ;Number of variables
PUSH@ ASCIIZ MsgInput   ;Input message string
PUSH WORD MsgInLength   ;Length of input message
PUSH@ OTHER DataArea    ;Updated message (returned)
PUSH WORD DataLength    ;Length of updated message buffer
PUSH@ WORD MsgLength    ;Length of updated message (returned)
CALL DosInsMessage

Returns WORD
```

# DosKillProcess – Terminate Process

This call flags a process to terminate and returns the termination code to its parent.

**DosKillProcess (ActionCode, ProcessID)**

## Parameters

**ActionCode** (*USHORT*) – input

The processes to be flagged for termination.

Value	Definition
0	A process and all its descendant processes. The process must be either the current process or a child process created by the current process. (Detached processes cannot be flagged for termination.) After the indicated process terminates, its descendants are flagged for termination.
1	Any process. Only the indicated process is flagged for termination.

**ProcessID** (*PID*) – input

Process ID of the process, or root process of the process tree to be flagged for termination.

**rc** (*USHORT*) – return

Return code descriptions are:

0	NO_ERROR
13	ERROR_INVALID_DATA
303	ERROR_INVALID_PROCID
305	ERROR_NOT_DESCENDANT

## Remarks

DosKillProcess allows a process to send the termination signal SIGTERM to another process or group of processes. The default action of the system is to terminate each of the processes. A process can intercept this action by installing a signal handler for SIGTERM with DosSetSigHandler. This gives the process the opportunity to clean up its files before it terminates with DosExit.

If there is no signal handler, the effect on the process is the same as if one of its threads had issued DosExit for the entire process. All file buffers are flushed and the handles opened by the process are closed. However, any internal buffers managed by programs external to OS/2 are not flushed. An example of such a buffer could be a C language library's internal character buffer.

If a parent process is waiting for a child process to end because of a DosCwait request, and the child is sent the SIGTERM signal but does not have a SIGTERM signal handler installed, the DosCwait request returns the "unintercepted DosKillProcess" termination code.

## C Language

```
#define INCL_DOSPROCESS
```

```
USHORT rc = DosKillProcess(ActionCode, ProcessID);
```

```
USHORT      ActionCode;    /* Indicate to flag descendant processes */
PID          ProcessID;    /* ID of process or root of process tree */

USHORT      rc;            /* return code */
```

# DosKillProcess — Terminate Process

## Assembler Language

```
EXTRN  DosKillProcess:FAR
INCL_DOSPROCESS    EQU 1

PUSH  WORD    ActionCode    ;Indicator for child process termination
PUSH  WORD    ProcessID     ;ID of the process being terminated
CALL  DosKillProcess
```

Returns WORD

## Example

The following example demonstrates how to create a process, obtain process ID information, and kill a process. Process1 invokes process2 to run asynchronously. It obtains and prints some PID information, and then kills process2.

## DosKillProcess — Terminate Process

```
/* ---- process1.c ---- */

#define INCL_DOSPROCESS

#include <os2.h>

#define START_PROGRAM "process2.exe" /* Program pointer */

main()
{
    CHAR          ObjFail [50];      /* Object name buffer */
    RESULTCODES   ReturnCodes;      /*
    PIDINFO       PidInfo;           */
    PID           ParentID;          /*
    USHORT        rc;

    printf("Process1 now running. \n");

    /** Start a child process. **/
    if(!DosExecPgm(ObjFail,          /* Object name buffer */
                  sizeof(ObjFail),  /* Length of obj. name buffer */
                  EXEC_ASYNC,       /* Execution flag - asynchronous */
                  NULL,             /* No args. to pass to process2 */
                  NULL,             /* Process2 inherits process1's environment */
                  &ReturnCodes,     /* Ptr. to resultcodes struct. */
                  START_PROGRAM)))  /* Name of program file */
        printf("Process2 started. \n");

    /** Obtain Process ID information and print it **/
    if(!rc=DosGetPID(&PidInfo))      /* Process ID's (returned) */
        printf("DosGetPID: current process ID is %d; thread ID is %d; parent process ID is %d.\n",
              PidInfo.pid, PidInfo.tid, PidInfo.pidParent);
    if(!rc=DosGetPPID(
        ReturnCodes.codeTerminate, /* Process whose parent is wanted */
        &ParentID))               /* Address to put parent's PID */
        printf("Child process ID is %d; Parent process ID is %d.\n",
              ReturnCodes.codeTerminate, ParentID);

    /** Terminate process2 **/
    if(!rc=DosKillProcess(DKP_PROCESSTREE, /* Action code - kill process and descendants */
                          ReturnCodes.codeTerminate)) /* PID of root of process tree */
        printf("Process2 terminated by process1.\n");
}

/* ---- process2.c ---- */

#define INCL_DOSPROCESS

#include <os2.h>

#define SLEEPTIME 500L
#define RETURN_CODE 0

main()
{
    printf("Process2 now running.\n");

    /** Sleep to allow process1 to kill it */
    DosSleep(SLEEPTIME);          /* Sleep interval */
    DosExit(EXIT_PROCESS,         /* Action Code */
            RETURN_CODE);         /* Result Code */
}
```

# DosLoadModule — Load Dynamic Link Module

---

This call loads a dynamic link module and returns a handle for the module.

<b>DosLoadModule</b> ( <i>ObjNameBuf</i> , <i>ObjNameBufL</i> , <i>ModuleName</i> , <i>ModuleHandle</i> )
---

## Parameters

**ObjNameBuf** (*PSZ*) — output

Address of the name of an object that contributed to the failure of **DosLoadModule**.

**ObjNameBufL** (*USHORT*) — input

Length, in bytes, of the buffer described by **ObjNameBuf**.

**ModuleName** (*PSZ*) — input

Address of the ASCIIZ name string containing the dynamic link module name or the pathname string. The filename extension used for dynamic link libraries is .DLL. When a pathname is provided, the module name may have any extension. The internal module name (the name given in the LIBRARY statement in the .DEF file) must be the same as the filename without the extension.

**ModuleHandle** (*PHMODULE*) — output

Address of the handle for the dynamic link module.

**rc** (*USHORT*) — return

Return code descriptions are:

0	NO_ERROR
2	ERROR_FILE_NOT_FOUND
3	ERROR_PATH_NOT_FOUND
4	ERROR_TOO_MANY_OPEN_FILES
5	ERROR_ACCESS_DENIED
8	ERROR_NOT_ENOUGH_MEMORY
11	ERROR_BAD_FORMAT
26	ERROR_NOT_DOS_DISK
32	ERROR_SHARING_VIOLATION
33	ERROR_LOCK_VIOLATION
36	ERROR_SHARING_BUFFER_EXCEEDED
95	ERROR_INTERRUPT
108	ERROR_DRIVE_LOCKED
123	ERROR_INVALID_NAME
127	ERROR_PROC_NOT_FOUND
180	ERROR_INVALID_SEGMENT_NUMBER
182	ERROR_INVALID_ORDINAL
190	ERROR_INVALID_MODULETYPE
191	ERROR_INVALID_EXE_SIGNATURE
192	ERROR_EXE_MARKED_INVALID
194	ERROR_ITERATED_DATA_EXCEEDS_64k
195	ERROR_INVALID_MINALLOCSIZE
196	ERROR_DYNLINK_FROM_INVALID_RING
198	ERROR_INVALID_SEGDPL
199	ERROR_AUTODATASEG_EXCEEDS_64k
201	ERROR_RELOC_CHAIN_XCEEDS_SEGLIM
206	ERROR_FILENAME_EXCED_RANGE

# DosLoadModule — Load Dynamic Link Module

## Remarks

If the file is an OS/2 dynamic link module, it is loaded, and a handle is returned. This handle is used for freeing the dynamic link module with a `DosFreeModule` request, getting procedure addresses with `DosGetProcAddr` requests, and getting the fully qualified file name with a `DosGetModName` request.

`DosLoadModule` may not be called from a dynamic library initialization routine if the module to be loaded or any module referenced by it contains a dynamic link library initialization routing.

## C Language

```
#define INCL_DOSMODULEMGR

USHORT rc = DosLoadModule(ObjNameBuf, ObjNameBufL, ModuleName, ModuleHandle);

PSZ      ObjNameBuf;    /* Address of object name buffer */
USHORT   ObjNameBufL;   /* Length of object name buffer */
PSZ      ModuleName;    /* Address of module name string */
PHMODULE ModuleHandle;  /* Address of module handle (returned) */

USHORT   rc;            /* return code */
```

## Assembler Language

```
EXTRN DosLoadModule:FAR
INCL_DOSMODULEMGR EQU 1

PUSH@ OTHER ObjNameBuf ;Object name buffer (returned)
PUSH WORD ObjNameBufL ;Length of object name buffer
PUSH@ ASCIIZ ModuleName ;Module name string
PUSH@ WORD ModuleHandle ;Module handle (returned)
CALL DosLoadModule
```

Returns WORD

## Example

This example loads a module.

```
#define INCL_DOSMODULEMGR

#define MODULE_NAME "abcd"
#define FULL_MODULE_NAME "\\nifty\\abcd.dll"

CHAR LoadError[100];
HMODULE ModuleHandle;
USHORT rc;

if (DosLoadModule(LoadError,          /* Object name buffer */
                 sizeof(LoadError),  /* Length of object name buffer */
                 MODULE_NAME,        /* Module name string */
                 &ModuleHandle) == 2) /* Module handle */
```

# DosLockSeg – Lock Segment in Memory

---

This call locks a discardable segment in memory.

<b>DosLockSeg (Selector)</b>
------------------------------

## Parameters

**Selector (SEL)** – input

Selector of the segment to be locked.

**rc (USHORT)** – return

Return code descriptions are:

0	NO_ERROR
5	ERROR_ACCESS_DENIED
157	ERROR_DISCARDED

## Remarks

Discardable segments are useful for holding objects that are accessed for short periods of time and that can be regenerated quickly if discarded. Examples are cache buffers for a data base package, saved bitmap images for obscured windows or precomputed display images for a word processing application.

Discardable memory is allocated by a call to DosAllocSeg or DosAllocHuge with AllocFlags bit 2 set. Upon allocation, the memory is automatically locked for access by the allocating process and cannot be discarded. After the segment is unlocked by a DosUnlockSeg request, the memory can be discarded by the memory manager to remedy a low memory situation. Once memory is discarded, a DosLockSeg call returns ERROR\_DISCARDED. The memory is reallocated by a call to DosReallocSeg or DosReallocHuge. The reallocation request automatically locks the memory.

Memory allocated as discardable by a DosAllocSeg or DosAllocHuge request may also have been designated as shareable. Sharing processes that access the discardable shared memory with DosGetSeg have to lock the memory by calling DosLockSeg. See DosGetSeg for more information.

DosLockSeg and DosUnlockSeg calls may be nested. If DosLockSeg is called multiple times to lock a segment, the same number of calls must be made to DosUnlockSeg before the segment is unlocked. However, if a segment is locked 255 times, it becomes permanently locked. Additional calls to DosLockSeg and DosUnlockSeg have no effect on the segment's locked state.

This function is used on segments that have been allocated through DosAllocSeg with AllocFlags bit 2 (0100B) set. It may be also used on segments that are non-discardable, in which case it has no effect.

## C Language

```
#define INCL_DOSMEMMGR
```

```
USHORT rc = DosLockSeg(Selector);
```

```
SEL Selector; /* Selector to lock */
```

```
USHORT rc; /* return code */
```

## Assembler Language

```
EXTRN DosLockSeg:FAR  
INCL_DOSMEMMGR EQU 1
```

```
PUSH WORD Selector ;Selector to lock  
CALL DosLockSeg
```

Returns WORD

# DosMakeNmPipe – Create Named Pipe

This call creates the specified named pipe and returns its handle.

<b>DosMakeNmPipe</b> ( <b>PipeName</b> , <b>PipeHandle</b> , <b>OpenMode</b> , <b>PipeMode</b> , <b>OutBufSize</b> , <b>InBufSize</b> , <b>TimeOut</b> )
--

## Parameters

**PipeName** (*PSZ*) – input

Address of the ASCIIZ name of the pipe to be opened. Pipes are named \PIPE\PipeName.

**PipeHandle** (*PHPIPE*) – output

Address of the handle of the named pipe that is created.

**OpenMode** (*USHORT*) – input

The OpenMode parameter contains the following bit fields:

Bit	Description
15	Reserved and must be zero.
14	Write-Through flag: The file is opened as follows:  0 = Write-behind to remote pipes is allowed.  1 = Write-behind to remote pipes is not allowed.  Setting the Write-Through flag is meaningful only for a remote pipe. Occasionally, data written to a remote pipe is buffered locally and then sent across the network to the pipe at a later time. Setting the Write-Through bit ensures that data is sent to the remote pipe as soon as it is written.
13 – 8	Reserved and must be zero.
7	Inheritance flag: The file handle is as follows:  0 = Pipe handle is inherited by a spawned process resulting from a DosExecPgm call.  1 = Pipe handle is private to the current process and cannot be inherited.
6 – 2	Reserved and must be zero.
1 – 0	Access field: The pipe access is assigned as follows:  00 = In-bound pipe (client to server)  01 = Out-bound pipe (server to client)  10 = Duplex pipe (server to/from client)  Any other value is invalid.

**PipeMode** (*USHORT*) – input

The PipeMode parameter contains the following bit fields:

Bit	Description
15	Blocking flag: The pipe is defined as follows:  0 = Reads/Writes block if no data available.  1 = Reads/Writes return immediately if no data available.  Reads normally block until at least partial data can be returned. Writes by default block until all bytes requested have been written. Non-blocking mode (1) changes this behavior as follows: <ul style="list-style-type: none"><li>• DosRead returns immediately with error NO_DATA if no data is available.</li></ul>



## DosMakeNmPipe — Create Named Pipe

- DosWrite returns immediately with BytesWritten = 0 if insufficient buffer space is available in the pipe or the entire data area is transferred.

**14 – 11** Reserved and must be zero.

**10** Type of named pipe: The pipe is defined as follows:

0 = Pipe is a byte stream pipe.

1 = Pipe is a message stream pipe.

All writes to message stream pipes record the length of the write along with the written data (see DosWrite). The first two bytes of each message represents the length of that message and is called the message header. A header of all zeros is reserved. Zero length messages are not allowed (OS/2 no-ops zero-length I/Os).

**9** Reserved and must be zero.

**8** Read mode: The pipe is defined as follows:

0 = Read pipe as a byte stream.

1 = Read pipe as a message stream.

Message pipes can be read as byte or message streams, depending on this bit. Byte pipes can only be read as byte streams (see DosRead)

**7 – 0** ICount field (Instance count): Byte wide (8-bit) count to control pipe instances. When making the first instance of a named pipe, ICount specifies how many instances can be created. Instances are as follows:

Value	Definition
<b>1</b>	This can be the only instance (pipe is unique).
<b>1 &lt; value &lt; 255</b>	The number of instances is limited to the value specified.
<b>-1</b>	The number of instances is unlimited.
<b>0</b>	Reserved value.

Subsequent attempts to make a pipe fails if the maximum number of allowed instances already exists. The ICount parameter is ignored when making any other than the first instance of a pipe. When multiple instances are allowed, multiple clients can simultaneously issue DosOpen to the same pipe name and get handles to distinct pipe instances.

**OutBufSize (USHORT) — input**

An advisory to the system of the number of bytes to allocate for the outgoing buffer.

**InBufSize (USHORT) — input**

An advisory to the system of the number of bytes to allocate for the incoming buffer.

**Timeout (ULONG) — Input**

Default value for the Timeout parameter to DosWaitNmPipe. This value may be set only at the creation of the first instance of the pipe name. If the value is zero, a system wide default value (50 ms) is chosen.

**rc (USHORT) — return**

Return code descriptions are:

<b>0</b>	<b>NO_ERROR</b>
<b>3</b>	<b>ERROR_PATH_NOT_FOUND</b>
<b>8</b>	<b>ERROR_NOT_ENOUGH_MEMORY</b>
<b>84</b>	<b>ERROR_OUT_OF_STRUCTURES</b>
<b>87</b>	<b>ERROR_INVALID_PARAMETER</b>
<b>231</b>	<b>ERROR_PIPE_BUSY</b>

# DosMakeNmPipe – Create Named Pipe

## Remarks

A named pipe provides two-way communication between a server process and a number of client processes. In addition, the named pipe can have multiple instances created by multiple server processes.

The server creates the pipe with `DosMakeNmPipe`. The `ICount` parameter is significant only for the first instance created of the named pipe. The ASCIIZ name string specified for the named pipe must include the prefix `\PIPE\`.

After creating the named pipe, the server issues `DosConnectNmPipe` to wait for a client to open the pipe with `DosOpen`. If all instances of a named pipe are busy, a client process can issue `DosWaitNmPipe` and wait for an instance to become available before it reissues `DosOpen`. A client can determine whether the pipe is ready to accept a `DosOpen` by issuing `DosPeekNmPipe` to return the pipe's state.

Server and client processes communicate by issuing `DosRead`, `DosReadAsync`, `DosWrite`, and `DosWriteAsync` calls. `DosBufReset` can be used to synchronize read and write dialogs. A server process that need to support a large number of clients for a local named pipe can coordinate access to the pipe with `DosSetNmPipeSem` and `DosQNmPipeSemState` calls.

Server and client processes can also communicate by means of transaction and procedure calls. `DosTransactNmPipe` and `DosCallNmPipe` are supported only for duplex message pipes.

Issuing `DosClose` ends the client's access to the named pipe. To prepare the pipe for its next client, the server process issues `DosDisconnectNmPipe` followed by `DosConnectNmPipe`.

When all handles to one end of the pipe are closed, the pipe is considered broken. If the pipe is broken and the server issues `DosClose`, the pipe is immediately deallocated.

## C Language

```
#define INCL_DOSNMPICES
```

```
USHORT rc = DosMakeNmPipe(PipeName, PipeHandle, OpenMode, PipeMode,  
                          OutBufSize, InBufSize, Timeout);
```

```
PSZ      PipeName;      /* Pipe name */  
PHPIPE   PipeHandle;     /* Pipe handle (returned) */  
USHORT   OpenMode;       /* DOS open mode of pipe */  
USHORT   PipeMode;       /* Pipe open mode */  
USHORT   OutBufSize;     /* Advisory outgoing buffer size */  
USHORT   InBufSize;      /* Advisory incoming buffer size */  
ULONG    Timeout;        /* Timeout for DosWaitNmPipe */
```

```
USHORT    rc;            /* return code */
```

## Assembler Language

```
EXTRN DosMakeNmPipe:FAR  
INCL_DOSNMPICES EQU 1
```

```
PUSH@ ASCIIZ PipeName ;Pipe name  
PUSH@ WORD PipeHandle ;Pipe handle (returned)  
PUSH WORD OpenMode ;DOS open mode of pipe  
PUSH WORD PipeMode ;Pipe open mode  
PUSH WORD OutBufSize ;Advisory outgoing buffer size  
PUSH WORD InBufSize ;Advisory incoming buffer size  
PUSH DWORD Timeout ;Timeout for DosWaitNmPipe  
CALL DosMakeNmPipe
```

Returns WORD

# DosMakePipe — Create Pipe

---

This call creates a pipe.

**DosMakePipe (ReadHandle, WriteHandle, PipeSize)**

## Parameters

**ReadHandle** (*PHFILE*) — output

Address of the read handle of the pipe.

**WriteHandle** (*PHFILE*) — output

Address of the write handle of the pipe.

**PipeSize** (*USHORT*) — input

Storage size, in bytes, to reserve for the pipe.

**rc** (*USHORT*) — return

Return code descriptions are:

0	NO_ERROR
4	ERROR_TOO_MANY_OPEN_FILES
8	ERROR_NOT_ENOUGH_MEMORY
109	ERROR_BROKEN_PIPE

## Remarks

Pipes are mechanisms used within a closely related group of processes. There are no control, permission mechanisms, or checks performed on operations to pipes.

When there is insufficient space in a pipe for the data being written, a requesting thread blocks until enough data is removed to allow the write request to be satisfied. If the PipeSize parameter is 0, then the pipe is created with a default size of 512 bytes.

When all users close the handles, a pipe is deleted. If two processes are communicating by a pipe and the process reading the pipe ends, the next write gets the "ERROR\_BROKEN\_PIPE" error code.

## C Language

```
#define INCL_DOSQUEUES

USHORT rc = DosMakePipe(ReadHandle, WriteHandle, PipeSize);

PHFILE      ReadHandle;    /* Address to put read handle (returned) */
PHFILE      WriteHandle;   /* Address to put write handle (returned) */
USHORT      PipeSize;      /* Size to reserve for the pipe */

USHORT      rc;            /* return code */
```

## Assembler Language

```
EXTRN DosMakePipe:FAR
INCL_DOSQUEUES EQU 1

PUSH@ WORD ReadHandle ;Read handle (returned)
PUSH@ WORD WriteHandle ;Write handle (returned)
PUSH WORD PipeSize ;Size to reserve for the pipe
CALL DosMakePipe

Returns WORD
```

# DosMemAvail – Get Size of Largest Free Memory Block

---

This call returns the size of the largest block of free memory.

<b>DosMemAvail (MemAvailSize)</b>
-----------------------------------

## Parameters

**MemAvailSize** (*PULONG*) – output

Address of the size of the largest free block of memory.

**rc** (*USHORT*) – return

Return code description is:

0            NO\_ERROR

## Remarks

DosMemAvail allows an application to determine how heavily used system memory is at a particular time. The returned value is a “snapshot” that may be valid only at the moment this function is issued and can be expected to change at any time due to system activity.

This call can be used as an indicator for memory availability before a call to DosAllocHuge is made.

## C Language

```
#define INCL_DOSMEMMGR

USHORT rc = DosMemAvail(MemAvailSize);

PULONG MemAvailSize; /* Size available (returned) */

USHORT rc; /* return code */
```

## Assembler Language

```
EXTRN DosMemAvail:FAR
INCL_DOSMEMMGR EQU 1

PUSH@ DWORD MemAvailSize ;Size available (returned)
CALL DosMemAvail
```

Returns WORD

---

This call creates a subdirectory.

<b>DosMkDir</b> (DirName, Reserved)
-------------------------------------

## Parameters

**DirName** (PSZ) — input

Address of the ASCIIZ directory path name, which may or may not include a drive specification. If no drive is specified, the current drive is assumed.

DosQSysInfo is called by an application during initialization to determine the maximum path length allowed by OS/2.

**Reserved** (ULONG) — input

Reserved and must be set to zero.

**rc** (USHORT) — return

Return code descriptions are:

0	NO_ERROR
3	ERROR_PATH_NOT_FOUND
5	ERROR_ACCESS_DENIED
26	ERROR_NOT_DOS_DISK
87	ERROR_INVALID_PARAMETER
108	ERROR_DRIVE_LOCKED
206	ERROR_FILENAME_EXCED_RANGE

## Remarks

If any subdirectory names in the path do not exist, the subdirectory is not created. Upon return, a subdirectory is created at the end of the specified path.

DosQSysInfo must be used by an application to determine the maximum path length supported by OS/2. The returned value should be used to dynamically allocate buffers that are to be used to store paths.

If a program running with the NEWFILES bit set tries to create a directory with blanks immediately preceding the dot on a FAT drive, the system rejects the name. For example, if c: is a FAT drive, the name "file .txt" is rejected and the name "file.txt" is accepted.

## C Language

```
#define INCL_DOSFILEMGR
```

```
USHORT rc = DosMkDir(DirName, Reserved);
```

```
PSZ      DirName;    /* New directory string name */
ULONG    0;          /* Reserved (must be zero) */
```

```
USHORT    rc;        /* return code */
```

## Assembler Language

```
EXTRN DosMkDir:FAR
INCL_DOSFILEMGR EQU 1
```

```
PUSH@ ASCIIZ DirName ;New directory name string
PUSH DWORD 0         ;Reserved (must be zero)
CALL DosMkDir
```

Returns WORD

## DosMkDir2 –

# Make Subdirectory and Define Extended Attributes

---

This call creates a subdirectory that has extended attributes associated with it.

**DosMkDir2** (DirName, EABuf, Reserved)

### Parameters

**DirName** (*PSZ*) – input

Address of the ASCIIZ directory path name, which may or may not contain a drive specification. If no drive is specified, the current drive is assumed.

DosQSysInfo is called by an application during initialization to determine the maximum path length allowed by OS/2.

**EABuf** (*PEAOP*) – input/output

Address of the extended attribute buffer, which contains an EAOP structure. An EAOP structure has the following format:

**fpGEAList** (*PGEALIST*)

Address of GEAList. GEAList is a packed array of variable length “get EA” structures, each containing an EA name and the length of the name.

**fpFEAList** (*PFEALIST*)

Address of FEAList. FEAList is a packed array of variable length “full EA” structures, each containing an EA name and its corresponding value, as well as the lengths of the name and the value.

**oError** (*ULONG*)

Offset into structure where error has occurred.

On input, the fpGEAList field and oError fields are ignored. The EA setting operation is performed on the information contained in FEAList. If extended attributes are not to be defined or modified, then EABuf must be set to null. Following is the FEAList format:

**cbList** (*ULONG*)

Length of the FEA list, including the length itself.

**list** (*FEA*)

List of FEA structures. An FEA structure has the following format:

**Flags** (*BYTE*)

Bit indicator describing the characteristics of the EA being defined.

Bit	Description
15	Critical EA.
14 – 0	Reserved and must be set to zero.

If bit 15 is set to 1, this indicates a critical EA. If bit 15 is 0, this means the EA is noncritical; that is, the EA is not essential to the intended use by an application of the file with which it is associated.

**cbName** (*BYTE*)

Length of EA ASCIIZ name, which does not include the null character.

**cbValue** (*USHORT*)

Length of EA value, which cannot exceed 64KB.

**szName** (*PSZ*)

Address of the ASCIIZ name of EA.

**aValue** (*PSZ*)

Address of the free-format value of EA.

**Note:** The szName and aValue fields are not included as part of header or include files. Because of their variable lengths, these entries must be built manually.

## Make Subdirectory and Define Extended Attributes

On output, fpGEAList is unchanged. fpFEAList is unchanged as is the area pointed to by fpFEAList. If an error occurred during the set, oError is the offset of the FEA where the error occurred. The API return code is the error code corresponding to the condition generating the error. If no error occurred, oError is undefined.

If EABuf is 0x00000000, then no extended attributes are defined for the directory.

**Reserved (ULONG)** — input

Reserved and must be set to zero.

**rc (USHORT)** — return

Return code descriptions are:

<b>0</b>	<b>NO_ERROR</b>
<b>3</b>	<b>ERROR_PATH_NOT_FOUND</b>
<b>5</b>	<b>ERROR_ACCESS_DENIED</b>
<b>26</b>	<b>ERROR_NOT_DOS_DISK</b>
<b>87</b>	<b>ERROR_INVALID_PARAMETER</b>
<b>108</b>	<b>ERROR_DRIVE_LOCKED</b>
<b>206</b>	<b>ERROR_FILENAME_EXCED_RANGE</b>
<b>254</b>	<b>ERROR_INVALID_EA_NAME</b>
<b>255</b>	<b>ERROR_EA_LIST_INCONSISTENT</b>

### Remarks

DosMkDir2 allows an application to define extended attributes for a subdirectory at the time of its creation.

If any subdirectory names in the path do not exist, the subdirectory is not created. Upon return, a subdirectory is created at the end of the specified path.

DosQSysInfo must be used by an application to determine the maximum path length supported by OS/2. The returned value should be used to dynamically allocate buffers that are to be used to store paths.

If a program running with the NEWFILES bit set tries to create a directory with blanks immediately preceding the dot on a FAT drive, the system rejects the name. For example, if c: is a FAT drive, the name "file .txt" is rejected and the name "file.txt" is accepted.

# Make Subdirectory and Define Extended Attributes

## C Language

```
typedef struct _GEA {          /* gea */

    BYTE cbName;               /* name length not including NULL */
    CHAR szName[1];           /* attribute name */

} GEA;

typedef struct _GEALIST {     /* geal */

    ULONG cbList;              /* total bytes of structure including full list */
    GEA list[1];              /* variable length GEA structures */

} GEALIST;

typedef struct _FEA {         /* fea */

    BYTE fEA;                  /* flags */
    BYTE cbName;               /* name length not including NULL */
    USHORT cbValue;            /* value length */

} FEA;

typedef struct _FEALIST {     /* feal */

    ULONG cbList;              /* total bytes of structure including full list */
    FEA list[1];              /* variable length FEA structures */

} FEALIST;

typedef struct _EAOP {        /* eaop */

    PGEALIST fpGEAList;        /* general EA list */
    PFEALIST fpFEAList;        /* full EA list */
    ULONG oError;

} EAOP;

#define INCL_DOSFILEMGR

USHORT rc = DosMkDir2(DirName, EABuf, Reserved);

PSZ      DirName;             /* New directory name string */
PEAOP    EABuf;               /* Extended attribute buffer */
ULONG    0;                   /* Reserved (must be zero) */

USHORT    rc;                  /* return code */
```



## Make Subdirectory and Define Extended Attributes

### Assembler Language

```

GEA    struc

    gea_cbName    db  ?           ;name length not including NULL
    gea_szName    db  1 dup (?)   ;attribute name

GEA    ends

GEALIST    struc

    geal_cbList    dd  ?           ;total bytes of structure including full list
    geal_list      db  size GEA * 1 dup (?) ;variable length GEA structures

GEALIST    ends

FEA    struc

    fea_fEA        db  ? ;flags
    fea_cbName     db  ? ;name length not including NULL
    fea_cbValue    dw  ? ;value length

FEA    ends

FEALIST    struc

    feal_cbList    dd  ?           ;total bytes of structure including full list
    feal_list      db  size FEA * 1 dup (?) ;variable length FEA structures

FEALIST    ends

EAOP    struc

    eaop_fpGEAList dd  ? ;general EA list
    eaop_fpFEAList dd  ? ;full EA list
    eaop_oError    dd  ? ;

EAOP    ends

EXTRN  DosMkDir2:FAR
INCL_DOSFILEMGR    EQU 1

PUSH@  ASCIIZ  DirName    ;New directory name string
PUSH@  OTHER  EABuf       ;Extended attribute buffer
PUSH    DWORD  0          ;Reserved (must be zero)
CALL    DosMkDir2

Returns WORD

```

# DosMonClose – Close Connection to Device Monitor

This call terminates character device monitoring. All monitor buffers associated with this process are flushed and closed.

<b>DosMonClose (Handle)</b>
-----------------------------

## Parameters

**Handle (HMONITOR)** – input

Device handle returned from a previous DosMonOpen call.

**rc (USHORT)** – return

Return code descriptions are:

<b>0</b>	<b>NO_ERROR</b>
<b>381</b>	<b>ERROR_MON_INVALID_HANDLE</b>

## Remarks

A single process may register one or more monitors with a character device using the same device handle returned from a previous DosMonOpen call. When DosMonClose is issued for a specific, opened device handle, all monitors for the current process registered with this handle terminate.

When DosMonClose is issued, the monitor loses access to the device data stream. Before issuing DosMonClose, monitor threads calling DosMonRead and DosMonWrite should be terminated. After DosMonClose has been called:

- DosMonRead calls return an ERROR\_MON\_BUFFER\_EMPTY return code.
- DosMonWrite calls return an ERROR\_NOT\_ENOUGH\_MEMORY return code.

Data segments containing monitor buffers should not be freed until after DosMonClose is called. If data segments containing monitor buffers are freed before DosMonClose is called, a GP fault occurs when DosMonClose is called and the process is terminated.

For a detailed description of this call see the chapter "Character Device Monitors" in the *IBM Operating System/2 Version 1.2 I/O Subsystems And Device Support Volume 1*.

## C Language

```
#define INCL_DOSMONITORS

USHORT rc = DosMonClose(Handle);

HMONITOR Handle; /* Monitor handle */

USHORT rc; /* return code */
```

## Assembler Language

```
EXTRN DosMonClose:FAR
INCL_DOSMONITORS EQU 1

PUSH WORD Handle ;Monitor handle
CALL DosMonClose

Returns WORD
```

# DosMonOpen —

## Open Connection to Device Monitor

xPM

---

This call gains access to a character device data stream.

<b>DosMonOpen (Devname, Handle)</b>
-------------------------------------

### Parameters

**Devname (PSZ)** — input

Address of the device name string.

**Handle (PHMONITOR)** — output

Address of the handle for the monitor. This value must be passed to DosMonReg to communicate with the device, and is passed to DosMonClose to close the connection to the monitor.

**rc (USHORT)** — return

Return code descriptions are:

<b>0</b>	<b>NO_ERROR</b>
<b>110</b>	<b>ERROR_OPEN_FAILED</b>
<b>379</b>	<b>ERROR_MON_INVALID_PARMS</b>
<b>380</b>	<b>ERROR_MON_INVALID_DEVNAME</b>

### Remarks

For a detailed description of this call see the chapter “Character Device Monitors” in the *IBM Operating System/2 Version 1.2 I/O Subsystems And Device Support Volume 1*.

Only one DosMonOpen call is necessary per device per process. That is, several DosMonReg calls can be made using the same monitor handle to the same device. This allows monitors to be registered using different values for Index from the same process and going to the same device. When the DosMonClose is issued, all of the monitors registered on the handle is closed.

### C Language

```
#define INCL_DOSMONITORS
```

```
USHORT rc = DosMonOpen(Devname, Handle);
```

```
PSZ      Devname;    /* Device name string */
PHMONITOR Handle;    /* Monitor handle (returned) */

USHORT   rc;         /* return code */
```

### Assembler Language

```
EXTRN DosMonOpen:FAR
INCL_DOSMONITORS EQU 1
```

```
PUSH@ ASCIIZ Devname    ;Device name string
PUSH@ WORD   Handle     ;Monitor handle (returned)
CALL  DosMonOpen
```

Returns WORD

# DosMonRead – Read Input from Monitor Structure

This call waits for and moves a data record from the input buffer of a registered character device monitor and places it in a private data area where the monitor can freely access it.

**DosMonRead (BufferI, WaitFlag, DataBuffer, Bytecnt)**

## Parameters

**BufferI (PBYTE)** – input

Address of the monitor input buffer.

**WaitFlag (UCHAR)** – input

Valid values are:

Value	Definition
0	The monitor thread that issues DosMonRead wishes to block until a data record is available in its input buffer.
1	The monitor thread that issues DosMonRead does not wish to block when its input buffer is empty.

**DataBuffer (PBYTE)** – input

Address of the buffer in the calling process address space that the data from the monitor's input buffer is read into. The length of DataBuffer must be the entry value of Bytecnt.

**Bytecnt (PUSHORT)** – input/output

Address of the length of DataBuffer, on entry to DosMonRead. On the return from DosMonRead, Bytecnt specifies the number of bytes of data moved.

**rc (USHORT)** – return

Return code descriptions are:

0	NO_ERROR
379	ERROR_MON_INVALID_PARMS
382	ERROR_MON_BUFFER_TOO_SMALL
383	ERROR_MON_BUFFER_EMPTY

## Remarks

For a detailed description of this call see the chapter "Character Device Monitors" in the *IBM Operating System/2 Version 1.2 I/O Subsystems And Device Support Volume 1*.

## C Language

```
#define INCL_DOSMONITORS
```

```
USHORT rc = DosMonRead(BufferI, WaitFlag, DataBuffer, Bytecnt);
```

```
PBYTE      BufferI;      /* Monitor input buffer */
UCHAR      WaitFlag;    /* Block/Run indicator */
PBYTE      DataBuffer;  /* Buffer into which records are read */
PUSHORT    Bytecnt;     /* Input/output parm-#bytes (returned) */

USHORT     rc;          /* return code */
```

1

# DosMonRead —

## Read Input from Monitor Structure

xPM

### Assembler Language

```
EXTRN DosMonRead:FAR
INCL_DOSMONITORS    EQU 1

PUSH@ OTHER    BufferI      ;Monitor input buffer
PUSH  WORD     WaitFlag    ;Block/Run indicator
PUSH@ OTHER    DataBuffer  ;Buffer into which records are read
PUSH@ WORD     Bytecnt     ;Input/output parm-#bytes (returned)
CALL  DosMonRead
```

Returns WORD

## DosMonReg – Register Set of Buffers as Monitor

This call establishes an input and output buffer structure to monitor an I/O stream for a character device.

**DosMonReg (Handle, BufferI, BufferO, Posflag, Index)**

### Parameters

**Handle (HMONITOR)** – input

Device handle returned from a previous DosMonOpen call.

**BufferI (PBYTE)** – input

Address of the monitor's input buffer. The monitor dispatcher moves data records into this buffer from the device driver (if the monitor is the first monitor in the monitor chain) or from the previous monitor, if any, in the monitor chain. The monitor takes data from this buffer for filtering by calling DosMonRead.

**BufferO (PBYTE)** – input

Address of the monitor's output buffer. The monitor places filtered data into this buffer by calling DosMonWrite. The monitor dispatcher moves data records from this buffer to the device driver (if the monitor is the last monitor in the monitor chain) or to the next monitor, if any, in the monitor chain.

**Posflag (USHORT)** – input

Used to specify the placement of a monitor's buffers with the monitor chain (FIRST, LAST or DEFAULT) and whether one or two threads are created by the monitor dispatcher to handle data movement.

Value	Definition
0	DEFAULT (no position preference) and one thread for data movement.
1	FIRST (monitor placed at beginning of monitor chain) and one thread for data movement.
2	LAST (monitor placed at end of monitor chain) and one thread for data movement.
3	DEFAULT with two threads for data movement.
4	FIRST with two threads for data movement.
5	LAST with two threads for data movement.

The first monitor in a monitor chain that registers as FIRST is placed at the head of the monitor chain. The next monitor that registers as FIRST follows the last monitor registered as FIRST, and so on. Similarly, the first monitor that registers as LAST is placed at the end of the monitor chain. The next monitor that registers as LAST is placed before the last monitor that registered as LAST, and so on. The first monitor that registers as DEFAULT is placed before the last monitor, if any, that registered as LAST. The next monitor that registers as DEFAULT is placed before the last monitor that registered as DEFAULT, and so on.

**Index (USHORT)** – input

Device specific value. For the keyboard it pertains to the session you wish to register a monitor on. For the printer it pertains to the data or code page monitor chain.

**rc (USHORT)** – return

Return code descriptions are:

0	NO_ERROR
8	ERROR_NOT_ENOUGH_MEMORY
165	ERROR_MONITORS_NOT_SUPPORTED
379	ERROR_MON_INVALID_PARMS
381	ERROR_MON_INVALID_HANDLE
382	ERROR_MON_BUFFER_TOO_SMALL

### Remarks

For a detailed description of this call see the chapter "Character Device Monitors" in the *IBM Operating System/2 Version 1.2 I/O Subsystems And Device Support Volume 1*.

# DosMonReg —

## Register Set of Buffers as Monitor

xPM

### C Language

```
#define INCL_DOSMONITORS

USHORT rc = DosMonReg(Handle, BufferI, BufferO, Posflag, Index);

HMONITOR      Handle;      /* Monitor handle */
PBYTE         BufferI;      /* Input buffer */
PBYTE         BufferO;      /* Output buffer */
USHORT        Posflag;     /* Position flag */
USHORT        Index;       /* Index */

USHORT        rc;          /* return code */
```

### Assembler Language

```
EXTRN DosMonReg:FAR
INCL_DOSMONITORS EQU 1

PUSH WORD Handle ;Monitor handle
PUSH@ OTHER BufferI ;Input buffer
PUSH@ OTHER BufferO ;Output buffer
PUSH WORD Posflag ;Position flag
PUSH WORD Index ;Index
CALL DosMonReg

Returns WORD
```

This call moves a filtered data record from the monitor's private data area into the monitor's output buffer.

**DosMonWrite (BufferO, DataBuffer, Bytecnt)**

## Parameters

**BufferO (PBYTE)** — input

Address of the monitor output buffer.

**DataBuffer (PBYTE)** — input

Address of the monitor's private data area that contains a filtered data record of length Bytecnt. DosMonWrite moves this filtered data record into the monitor's output buffer.

**Bytecnt (USHORT)** — input

Number of bytes in the data record.

**rc (USHORT)** — return

Return code descriptions are:

<b>0</b>	<b>NO_ERROR</b>
<b>8</b>	<b>ERROR_NOT_ENOUGH_MEMORY</b>
<b>379</b>	<b>ERROR_MON_INVALID_PARMS</b>
<b>384</b>	<b>ERROR_MON_DATA_TOO_LARGE</b>

## Remarks

For a detailed description of the use of this call see the chapter "Character Device Monitors" in the *IBM Operating System/2 Version 1.2 I/O Subsystems And Device Support Volume 1*.

## C Language

```
#define INCL_DOSMONITORS
```

```
USHORT rc = DosMonWrite(BufferO, DataBuffer, Bytecnt);
```

```
PBYTE      BufferO;      /* Monitor output buffer */
PBYTE      DataBuffer;   /* Buffer from which records are taken */
USHORT     Bytecnt;      /* Number of bytes */

USHORT     rc;           /* return code */
```

## Assembler Language

```
EXTRN DosMonWrite:FAR
INCL_DOSMONITORS EQU 1
```

```
PUSH@ OTHER BufferO      ;Monitor output buffer
PUSH@ OTHER DataBuffer   ;Buffer from which records are taken
PUSH WORD Bytecnt        ;Number of bytes
CALL DosMonWrite
```

Returns WORD



---

This call moves a file object to another location and changes its name.

<b>DosMove</b> ( <i>OldPathName</i> , <i>NewPathName</i> , <i>Reserved</i> )
--

### Parameters

**OldPathName** (*PSZ*) — input

Address of the old path name of the file to be moved.

**NewPathName** (*PSZ*) — input

Address of the new path name of the file.

**Reserved** (*ULONG*) — input

Reserved and must be set to zero.

**rc** (*USHORT*) — return

Return code descriptions are:

0	NO_ERROR
2	ERROR_FILE_NOT_FOUND
3	ERROR_PATH_NOT_FOUND
5	ERROR_ACCESS_DENIED
17	ERROR_NOT_SAME_DEVICE
26	ERROR_NOT_DOS_DISK
32	ERROR_SHARING_VIOLATION
36	ERROR_SHARING_BUFFER_EXCEEDED
87	ERROR_INVALID_PARAMETER
108	ERROR_DRIVE_LOCKED
206	ERROR_FILENAME_EXCED_RANGE
250	ERROR_CIRCULARITY_REQUESTED
251	ERROR_DIRECTORY_IN_CDS

### Remarks

This call is often used to change only the name of a file or subdirectory, allowing the file object to remain in the same subdirectory. Global file name characters are not allowed in the source or target name.

If the paths specified are different, this allows the subdirectory location of the file object to be changed as well. If a drive is specified for the target, it must be the same as the one specified or implied by the source.

Any attempts to move a parent subdirectory to one of its descendant subdirectories are rejected, because a subdirectory cannot be both an ancestor and a descendant of the the same subdirectory. Any attempts by a process to move the current subdirectory or any of its ancestors are also rejected.

Attributes (times and dates) of the source file object are moved to the target. If read-only files exist in the target path, they are not replaced.

DosQSysInfo is called during initialization by an application to determine the maximum path length allowed by OS/2.

DosMove can be used to change the case of a file on an FSD drive. The following example would change the name of the file to "File.Txt".

```
DosMove("file.txt", "File.Txt")
```

**Family API Considerations**

Some options operate differently in the DOS mode than in the OS/2 mode. Therefore, the following restriction applies to DosMove when coding for the DOS mode:

File names passed to OldPathName and NewPathName are truncated by the system in the DOS mode only. The application must truncate all files passed to OldPathName and NewPathName in the OS/2 mode or an error code is returned.

**C Language**

```
#define INCL_DOSFILEMGR

USHORT rc = DosMove(OldPathName, NewPathName, Reserved);

PSZ      OldPathName; /* Old path name string */
PSZ      NewPathName; /* New path name string */
ULONG    0;           /* Reserved (must be zero) */

USHORT    rc;          /* return code */
```

**Assembler Language**

```
EXTRN DosMove:FAR
INCL_DOSFILEMGR EQU 1

PUSH@ ASCIIZ OldPathName ;Old path name string
PUSH@ ASCIIZ NewPathName ;New path name string
PUSH DWORD 0 ;Reserved (must be zero)
CALL DosMove
```

Returns WORD

# DosMuxSemWait — Wait for One of N Semaphores to Clear

---

This call blocks a current thread until one of the specified semaphores is cleared.

<b>DosMuxSemWait</b> ( <i>IndexNbr</i> , <i>ListAddr</i> , <i>Timeout</i> )
---

## Parameters

**IndexNbr** (*PUSHORT*) — output

Address of the index number of the semaphore in the list of semaphores that satisfies the wait request.

**ListAddr** (*PVOID*) — input

Address of the structure for list of descriptors that define the semaphores to be waited on.

**semcount** (*USHORT*)

Number of semaphores.

**sementry** (*MUXSEM*)

Array of *MUXSEM* structures:

**reserved** (*USHORT*)

Reserved; must be zero.

**hsem** (*HSEM*)

Reference to the semaphore.

For a system semaphore, this reference is the handle returned by a *DosCreateSem* or *DosOpenSem* request that granted the requesting thread access to the semaphore.

For a RAM semaphore, this reference is the address of a doubleword of storage, allocated and initialized to zero by the application. This sets the semaphore as unowned. Other than initializing the doubleword to zero, an application must not modify a RAM semaphore directly; instead it manipulates the semaphore with semaphore function calls.

**Timeout** (*LONG*) — input

Action taken by the requesting thread when none of the semaphores in the list is available. The values that can be specified are:

Value	Definition
-1	The requesting thread waits indefinitely for a semaphore to be cleared.
0	The requesting thread returns immediately.
> 0	The requesting thread waits the indicated number of milliseconds for a semaphore to be cleared before resuming execution.

**rc** (*USHORT*) — return

Return code descriptions are:

0	NO_ERROR
95	ERROR_INTERRUPT
101	ERROR_EXCL_SEM_ALREADY_OWNED
121	ERROR_SEM_TIMEOUT
151	ERROR_INVALID_EVENT_COUNT
152	ERROR_TOO_MANY_MUXWAITERS
153	ERROR_INVALID_LIST_FORMAT

# DosMuxSemWait – Wait for One of N Semaphores to Clear

## Remarks

DosMuxSemWait checks a semaphore list. If any of the semaphores are clear, DosMuxSemWait returns. If all are set, DosMuxSemWait blocks until one of the semaphores is cleared or until the time out occurs.

Unlike other semaphore calls that block (DosSemRequest, DosSemWait, and DosSemSetWait), DosMuxSemWait is an edge-triggered, rather than a level-triggered, procedure. This means it returns whenever one of the semaphores on the list is cleared, regardless of how long that semaphore may remain clear. It is possible the semaphore may be reset before DosMuxSemWait returns. If a semaphore is cleared and then set prior to the thread's executing the DosMuxSemWait call, the event is lost. Events are only effective while a thread is in a DosMuxSemWait call.

This implementation allows DosMuxSemWait to be used in conjunction with one or more semaphores as a triggering or synchronizing device. One or more threads can call DosMuxSemWait for a particular semaphore. When an event occurs, another thread can clear the semaphore and then immediately set it again, arming it for the next event. Threads that were waiting on that semaphore return from DosMuxSemWait and resume execution.

## C Language

```
typedef struct _MUXSEMLIST {    /* mxsl */

    USHORT  cmxs;                /* count of MuxSem structures */
    MUXSEM  amxs[16];           /* MuxSem structure */

} MUXSEMLIST;

typedef struct _MUXSEM {        /* mxs */

    USHORT  zero;               /* zero */
    HSEM     hsem;              /* semaphore handle */

} MUXSEM;

#define INCL_DOSSEMAPHORES

USHORT  rc = DosMuxSemWait(IndexNbr, ListAddr, Timeout);

PUSHORT      IndexNbr;        /* Index number of event (returned) */
PVOID        ListAddr;        /* Semaphore list */
LONG         Timeout;         /* Timeout (in milliseconds) */

USHORT       rc;              /* return code */
```

## Assembler Language

```
MUXSEMLIST struc

    mxsl_cmxs dw ? ;count of MuxSem structures
    mxsl_amxs dw (size MUXSEM)/2 * 16 dup (?) ;MuxSem structure

MUXSEMLIST ends

MUXSEM struc

    mxs_zero dw ? ;zero
    mxs_hsem dd ? ;semaphore handle

MUXSEM ends

EXTRN DosMuxSemWait:FAR
INCL_DOSSEMAPHORES EQU 1

PUSH@ WORD    IndexNbr        ;Index number of event (returned)
PUSH@ OTHER   ListAddr        ;Semaphore list
PUSH  DWORD   Timeout         ;Timeout (in milliseconds)
CALL  DosMuxSemWait

Returns WORD
```

# DosMuxSemWait —

## Wait for One of N Semaphores to Clear

### Example

The following example illustrates notification of events between threads of the same process. The main thread sets two RAM semaphores and invokes two threads, each of which clears one of the semaphores. Meanwhile, the main thread waits for either of the two semaphores to clear, and then resumes execution, indicating the thread that notified first.

```
#define INCL_DOSPROCESS
#define INCL_DOSSEMAPHORES

#include<os2.h>
#include<stdio.h>

#define NUM_SEMS      2      /* Number of semaphores to wait on */
#define TIMEOUT       2000L  /* Timeout period */
#define SLEEPTIME     1000L  /* Sleep period for Thread_1 */
#define RETURN_CODE   0      /* Return Code for thread termination */

ULONG      RamSem1 = 0L;      /* Allocation and initialization of */
ULONG      RamSem2 = 0L;      /* two RAM semaphores */
ULONG far  *RamSem1Handle = &RamSem1; /* Semaphore handles */
ULONG far  *RamSem2Handle = &RamSem2;
TID        ThreadID[2];      /* Thread Identification */
BYTE       ThreadStack1[4000]; /* Thread Stack Area */
BYTE       ThreadStack2[4000];
MUXSEMLIST SemList;          /* Semaphore list structure */
USHORT     IndexNbr;          /* Index number for DosMuxSemWait */
USHORT     rc;                /* Return Code */

VOID APIENTRY Thread_1()
{
    USHORT r;

    printf("Begin Thread_1. It will sleep for 1 second. \n");

    /* Give Thread_2 a chance to execute */
    DosSleep(SLEEPTIME); /* Sleep Interval */
    if(!(r=DosSemClear(RamSem1Handle))) /* Semaphore handle */
        printf("RamSem1 cleared. \n");
    DosExit(EXIT_THREAD, /* Action Code */
            RETURN_CODE); /* Result Code */
}

VOID APIENTRY Thread_2()
{
    USHORT r;

    printf("Begin Thread_2. It will try to clear RamSem2 now. \n");
    if(!(r=DosSemClear(RamSem2Handle))) /* Semaphore Handle */
        printf("RamSem2 cleared. \n");
    DosExit(EXIT_THREAD, /* Action Code */
            RETURN_CODE); /* Result Code */
}
```

## DosMuxSemWait – Wait for One of N Semaphores to Clear

```
main()
{
    USHORT rc;                                /* Return Code */

    /* Set both semaphores */
    if(!rc=DosSemSet(RamSem1Handle)) /* Semaphore Handle */
        printf("RamSem1 set. \n");
    if(!rc=DosSemSet(RamSem2Handle)) /* Semaphore Handle */
        printf("RamSem2 set. \n");

    /* Initialize Semaphore list structure */
    SemList.cmxs = NUM_SEMS; /* Number of semaphores */
    SemList.amxs[0].zero = 0; /* This field must be 0 */
    SemList.amxs[1].zero = 0; /* This field must be 0 */
    SemList.amxs[0].hsem = RamSem1Handle; /* Semaphore handle */
    SemList.amxs[1].hsem = RamSem2Handle; /* Semaphore handle */

    /* Start the two threads */
    if(!DosCreateThread((PFNTHREAD) Thread_1, /* Thread address */
                       &ThreadID[0], /* Thread ID (returned) */
                       &ThreadStack1[3999])) /* End of thread stack */
        printf("Thread_1 created. \n");
    if(!DosCreateThread((PFNTHREAD) Thread_2, /* Thread address */
                       &ThreadID[1], /* Thread ID (returned) */
                       &ThreadStack2[3999])) /* End of thread stack */
        printf("Thread_2 created. \n");

    /* Wait for either semaphore to clear; then, indicate which */
    /* thread notified first. */

    if(!DosMuxSemWait(&IndexNbr, /* Index of semaphore */
                     /* cleared (returned) */
                     &SemList, /* Address of sem. list structure */
                     TIMEOUT)) /* Timeout period */
        printf("Back to main thread; semaphore cleared by Thread%d.\n",
              IndexNbr + 1);
}
}
```

# DosNewSize — Change File Size

FAPI

This call changes the size of a file.

**DosNewSize (FileHandle, FileSize)**

## Parameters

**FileHandle** (*HFILE*) — input

Handle of the file whose size is being changed.

**FileSize** (*ULONG*) — input

File's new size in bytes.

**rc** (*USHORT*) — return

Return code descriptions are:

0	NO_ERROR
5	ERROR_ACCESS_DENIED
6	ERROR_INVALID_HANDLE
26	ERROR_NOT_DOS_DISK
33	ERROR_LOCK_VIOLATION
87	ERROR_INVALID_PARAMETER
112	ERROR_DISK_FULL

## Remarks

When DosNewSize is called, the file must be open in a mode that allows write access. If the file is a read-only file, its read-only status must be changed with DosSetFileMode before you can open the file for write access.

The open file can be truncated or extended in size. If the file is being extended, the file system makes a reasonable attempt to allocate the additional bytes for the file in a contiguous (or nearly contiguous) space on the medium. The value of the new bytes is undefined.

## C Language

```
#define INCL_DOSFILEMGR

USHORT rc = DosNewSize(FileHandle, FileSize);

HFILE      FileHandle;    /* File handle */
ULONG      FileSize;      /* File's new size */

USHORT      rc;           /* return code */
```

## Assembler Language

```
EXTRN DosNewSize:FAR
INCL_DOSFILEMGR EQU 1

PUSH WORD FileHandle ;File handle
PUSH DWORD FileSize ;File's new size
CALL DosNewSize
```

Returns WORD

This call opens a new file, an existing file, a replacement for an existing file, a named pipe, or a device.

**DosOpen (FileName, FileHandle, ActionTaken, FileSize, FileAttribute, OpenFlag, OpenMode, Reserved)**

## Parameters

**FileName (PSZ)** – input

Address of the ASCII path name of the file, named pipe, or device to be opened.

**FileHandle (PHFILE)** – output

Address of the handle for the file, named pipe, or device.

**ActionTaken (PUSHORT)** – output

Address of the action taken as a result of DosOpen.

Value	Definition
0001H	File exists
0002H	File created
0003H	File replaced.

**FileSize (ULONG)** – input

File's new logical size (EOD), in bytes. This parameter is significant only when creating a new file or replacing an existing file. Otherwise, it is ignored.

**FileAttribute (USHORT)** – input

File attribute bits. Defined below:

Bit	Description
15 – 6	Reserved and must be zero.
5	File archive
4	Subdirectory
3	Reserved and must be zero.
2	System file
1	Hidden file
0	Read only file

These bits may be set individually or in combination. For example, an attribute value of 0021H (bits 5 and 0 set to 1) indicates a read-only file that should be archived.

**OpenFlag (USHORT)** – input

One-word field indicates the action to be taken if the file exists or does not exist.

Bit	Description
15 – 8	Reserved and must be zero.
7 – 4	0000 = Fail if file does not exist. 0001 = Create file if file does not exist.
3 – 0	0000 = Fail if the file already exists. 0001 = Open the file if it already exists. 0010 = Replace the file if it already exists.



## OpenMode (USHORT) — input

The OpenMode parameter contains the following bit flags:

Bit	Description										
15	<p>DASD Open flag:</p> <p>0 = FileName represents a file to be opened in the normal way.</p> <p>1 = FileName is "Drive:" and represents a mounted disk or diskette volume to be opened for direct access.</p>										
14	<p>Write-Through flag:</p> <p>0 = Writes to the file may be run through the file system buffer cache.</p> <p>1 = Writes to the file may go through the file system buffer cache but the sectors are written (actual file I/O completed) before a synchronous write call returns. This state of the file defines it as a synchronous file. For synchronous files, this is a mandatory bit in that the data must be written out to the medium for synchronous write operations.</p> <p>This bit is not inherited by child processes.</p>										
13	<p>Fail-Errors flag. Media I/O errors are handled as follows:</p> <p>0 = Reported through the system critical error handler.</p> <p>1 = Reported directly to the caller by way of return code.</p> <p>Media I/O errors generated through an IOCTL Category 8 function always get reported directly to the caller by way of return code. The Fail-Errors function applies only to non-IOCTL handle-based file I/O calls.</p> <p>This bit is not inherited by child processes.</p>										
12	<p>No-Cache/Cache flag:</p> <p>0 = It is advisable for the disk driver to cache the data in I/O operations on this file.</p> <p>1 = I/O to the file need not be done through the disk driver cache.</p> <p>This bit advises FSDs and device drivers whether it is worth caching the data. Like the write-through bit, this is a per-handle bit and is not inherited by child processes.</p>										
11	Reserved and must be zero.										
10 – 8	<p>The locality of reference flags contain information about how the application is to access the file.</p> <table> <tr> <th>Value</th><th>Definition</th></tr> <tr> <td>000</td><td>No locality known.</td></tr> <tr> <td>001</td><td>Mainly sequential access.</td></tr> <tr> <td>010</td><td>Mainly random access.</td></tr> <tr> <td>011</td><td>Random with some locality.</td></tr> </table>	Value	Definition	000	No locality known.	001	Mainly sequential access.	010	Mainly random access.	011	Random with some locality.
Value	Definition										
000	No locality known.										
001	Mainly sequential access.										
010	Mainly random access.										
011	Random with some locality.										
7	<p>Inheritance flag:</p> <p>0 = File handle is inherited by a spawned process resulting from a DosExecPgm call.</p> <p>1 = File handle is private to the current process.</p> <p>This bit is not inherited by child processes.</p>										
6 – 4	<p>Sharing Mode flags. This field defines any restrictions to file access placed by the caller on other processes:</p> <table> <tr> <th>Value</th><th>Definition</th></tr> <tr> <td>001</td><td>Deny Read/Write access</td></tr> <tr> <td>010</td><td>Deny Write access</td></tr> <tr> <td>011</td><td>Deny Read access</td></tr> <tr> <td>100</td><td>Deny neither Read or Write access (Deny None).</td></tr> </table> <p>Any other value is invalid.</p>	Value	Definition	001	Deny Read/Write access	010	Deny Write access	011	Deny Read access	100	Deny neither Read or Write access (Deny None).
Value	Definition										
001	Deny Read/Write access										
010	Deny Write access										
011	Deny Read access										
100	Deny neither Read or Write access (Deny None).										

**3** Reserved and must be zero.

**2 – 0** Access Mode flags. This field defines file access required by the caller:

Value	Definition
<b>000</b>	Read – Only access
<b>001</b>	Write – Only access
<b>010</b>	Read/Write access.

Any other value is invalid.

Any other combinations are invalid.

File sharing requires the cooperation of sharing processes. This cooperation is communicated through sharing and access modes. Any sharing restrictions placed on a file opened by a process are removed when the process closes the file with a DosClose request.

## Sharing Mode

Specify the type of access other processes may have to the file (sharing mode). For example, if it is permissible for other processes to continue reading the file while your process is operating on it, specify Deny Write. This sharing mode prevents other processes from writing to the file but still allows them to read it.

## Access Mode

Specify the type of access to the file needed by your process (access mode). For example, if your process requires Read/Write access, and another process has already opened the file with a sharing mode of Deny None, your DosOpen request succeeds. However, if the file is open with a sharing mode of Deny Write, your process is denied access.

If the file is inherited by a child process, all sharing and access restrictions are also inherited.

If an open file handle is duplicated by a call to DosDupHandle, all sharing and access restrictions are also duplicated.

**Reserved (ULONG) – input**

Reserved and must be set to zero.

**rc (USHORT) – return**

Return code descriptions are:

<b>0</b>	<b>NO_ERROR</b>
<b>2</b>	<b>ERROR_FILE_NOT_FOUND</b>
<b>3</b>	<b>ERROR_PATH_NOT_FOUND</b>
<b>4</b>	<b>ERROR_TOO_MANY_OPEN_FILES</b>
<b>5</b>	<b>ERROR_ACCESS_DENIED</b>
<b>12</b>	<b>ERROR_INVALID_ACCESS</b>
<b>26</b>	<b>ERROR_NOT_DOS_DISK</b>
<b>32</b>	<b>ERROR_SHARING_VIOLATION</b>
<b>36</b>	<b>ERROR_SHARING_BUFFER_EXCEEDED</b>
<b>82</b>	<b>ERROR_CANNOT_MAKE</b>
<b>87</b>	<b>ERROR_INVALID_PARAMETER</b>
<b>108</b>	<b>ERROR_DRIVE_LOCKED</b>
<b>110</b>	<b>ERROR_OPEN_FAILED</b>
<b>112</b>	<b>ERROR_DISK_FULL</b>
<b>206</b>	<b>ERROR_FILENAME_EXCED_RANGE</b>
<b>231</b>	<b>ERROR_PIPE_BUSY</b>
<b>99</b>	<b>ERROR_DEVICE_IN_USE</b>

## Remarks

A successful DosOpen request for a file returns a handle to access the file. The read/write pointer is set at the first byte of the file. The pointer's position may be changed by a DosChgFilePtr request or by read and write operations on the file.

The file's date and time can be queried by calling DosQFileInfo, and set by calling DosSetFileInfo.

FileAttribute sets attribute bits for the file object. Attributes of an existing file can be queried and set by DosQFileMode and DosSetFileMode. A file's read-only attribute may also be set with the OS/2 ATTRIB command.

FileAttribute cannot be set to Volume Label. Volume labels cannot be opened. DosSetFSInfo may be issued with a logical drive number to set volume label information.

The FileSize parameter affects the size of the file only when the file is a new file or a replacement for an existing one. If an existing file is simply opened, FileSize is ignored. DosNewSize may be called to change the existing file's size.

The value in FileSize is a recommended size for the file. If allocation of the full size fails, the open may still succeed. The file system makes a reasonable attempt to allocate the new size in as nearly contiguous an area as possible on the medium. When the file size is extended, the value of the new bytes is undefined.

The DASD Open bit provides direct access to an entire disk or diskette volume, independent of the file system. This mode of opening the volume currently mounted on the drive returns a handle to the caller, which represents the logical volume as a single file. The caller specifies this handle with a DosDevIOCtl Category 8 Function 0 request to block other processes from accessing the logical volume.

The file handle state bits can be set by the DosOpen and DosSetFHandState requests. An application can query the file handle state bits as well as the rest of the Open Mode field, by calling DosQFHandState. If a program running with the NEWFILES bit set tries to create or replace a file with blanks immediately preceding the dot on a FAT drive, the system rejects the name. For example, if c: is a FAT drive, the name "file .txt" is rejected and the name "file.txt" is accepted.

## Family API Considerations

Some options operate differently in the DOS mode than in the OS/2 mode. Therefore, the following restrictions apply to DosOpen when coding for the DOS mode:

- OpenMode restrictions:
  - Handles returned in response to DASD open are valid only for DosDevIOCtl.
  - Inheritance flag is not supported in DOS 2.X.
  - Write-Through flag must be set to zero.
  - Fail-errors flag must be set to zero.
  - Sharing mode field has meaning only if Sharing is loaded in DOS 3.X, ignored if Sharing is not loaded. Sharing mode is not supported in DOS 2.X.
  - Access mode field has meaning only if Sharing is loaded in DOS 3.X, ignored if Sharing is not loaded. Access mode field is not supported in DOS 2.X.
- Access mode is valid only if Sharing is loaded.

## Named Pipe Considerations

DosOpen opens the client end of a pipe by name and returns a handle. The open succeeds only if the pipe is in a listening state; otherwise, the open returns with ERROR\_PIPE\_BUSY. The pipe can be busy because of the following reasons:

- All instances of the pipe are already open.
- The pipe is closed but is not yet disconnected by the serving end.
- No DosConnectNmPipe is issued against the pipe after it is disconnected.

Once a given instance has been opened by a client, that same instance cannot be opened by another client at the same time. Pipes can only be two-ended; however, the opening process can duplicate the open handle as many times as desired.

Pipes are always opened with the pipe-specific states set to B = 0 (to block reads/writes) and RR = 00 (read the pipe as a byte stream). The client can change these modes by calling DosSetNmPHandState if desired.

The access and sharing modes specified on the open must be consistent with those specified on the DosMakeNmPipe request.

## C Language

```
#define INCL_DOSFILEMGR

USHORT rc = DosOpen(FileName, FileHandle, ActionTaken, FileSize,
                    FileAttribute, OpenFlag, OpenMode, Reserved);

PSZ      FileName;      /* File path name string */
PHFILE   FileHandle;     /* File handle (returned) */
PUSHORT  ActionTaken;    /* Action taken (returned) */
ULONG    FileSize;       /* File primary allocation */
USHORT   FileAttribute;  /* File Attribute */
USHORT   OpenFlag;       /* Open function type */
USHORT   OpenMode;       /* Open mode of the file */
ULONG    0;              /* Reserved (must be zero) */

USHORT    rc;            /* return code */
```

## Assembler Language

```
EXTRN DosOpen:FAR
INCL_DOSFILEMGR EQU 1

PUSH@ ASCIIZ FileName ;File path name string
PUSH@ WORD FileHandle ;File handle (returned)
PUSH@ WORD ActionTaken ;Action taken (returned)
PUSH DWORD FileSize ;File primary allocation
PUSH WORD FileAttribute ;File Attribute
PUSH WORD OpenFlag ;Open function type
PUSH WORD OpenMode ;Open mode of the file
PUSH DWORD 0 ;Reserved (must be zero)
CALL DosOpen

Returns WORD
```

# DosOpen —

## Open File

FAP1

### Example

This example opens a file.

```
#define INCL_DOSFILEMGR

#define OPEN_FILE 0x01
#define CREATE_FILE 0x10
#define FILE_ARCHIVE 0x20
#define FILE_EXISTS OPEN_FILE
#define FILE_NOEXISTS CREATE_FILE
#define DASD_FLAG 0
#define INHERIT 0x80
#define WRITE_THRU 0
#define FAIL_FLAG 0
#define SHARE_FLAG 0x10
#define ACCESS_FLAG 0x02

#define FILE_NAME "test.dat"
#define FILE_SIZE 800L
#define FILE_ATTRIBUTE FILE_ARCHIVE
#define RESERVED 0L

HFILE FileHandle;
USHORT Wrote;
USHORT Action;
PSZ FileData[100];
USHORT rc;

Action = 2;
strcpy(FileData, "Data...");
rc = DosOpen(FILE_NAME, /* File path name */
             &FileHandle, /* File handle */
             &Action, /* Action taken */
             FILE_SIZE, /* File primary allocation */
             FILE_ATTRIBUTE, /* File attribute */
             FILE_EXISTS | FILE_NOEXISTS, /* Open function type */
             DASD_FLAG | INHERIT, /* Open mode of the file */
             WRITE_THRU | FAIL_FLAG |
             SHARE_FLAG | ACCESS_FLAG,
             RESERVED); /* Reserved (must be zero) */
```

This call opens a new file, an existing file, or a replacement for an existing file. The opened file can have extended attributes.

**DosOpen2 (FileName, FileHandle, ActionTaken, FileSize, FileAttribute, OpenFlag, OpenMode, EABuf, Reserved)**

## Parameters

**FileName (PSZ)** – input

Address of the ASCIIZ path name of the file to be opened.

**FileHandle (PHFILE)** – output

Address of the handle for the file.

**ActionTaken (PUSHORT)** – output

Address of the action taken as a result of DosOpen2.

Value	Definition
0001H	File exists
0002H	File created
0003H	File replaced.

**FileSize (ULONG)** – input

File's new logical size (EOD), in bytes. This parameter is significant only when creating a new file or replacing an existing file. Otherwise, it is ignored.

**FileAttribute (USHORT)** – input

File attribute bits. Defined below:

Bit	Description
15 – 6	Reserved and must be zero.
5	File archive
4	Subdirectory
3	Reserved and must be zero.
2	System file
1	Hidden file
0	Read only file

These bits may be set individually or in combination. For example, an attribute value of 0021H (bits 5 and 0 set to 1) indicates a read-only file that should be archived.

**OpenFlag (USHORT)** – input

One-word field indicates the action to be taken if the file exists or does not exist.

Bit	Description
15 – 8	Reserved and must be zero.
7 – 4	0000 = Fail if file does not exist. 0001 = Create file if file does not exist.
3 – 0	0000 = Fail if the file already exists. 0001 = Open the file if it already exists. 0010 = Replace the file if it already exists.

**OpenMode (ULONG) — input**

The OpenMode parameter contains the following bit flags:

Bit	Description										
15	DASD Open flag:  0 = FileName represents a file to be opened in the normal way.  1 = FileName is "Drive:" and represents a mounted disk or diskette volume to be opened for direct access.										
14	Write-Through flag:  0 = Writes to the file may be run through the file system buffer cache.  1 = Writes to the file may go through the file system buffer cache but the sectors are written (actual file I/O completed) before a synchronous write call returns. This state of the file defines it as a synchronous file. For synchronous files, this is a mandatory bit in that the data must be written out to the medium for synchronous write operations.  This bit is not inherited by child processes.										
13	Fail-Errors flag. Media I/O errors are handled as follows:  0 = Reported through the system critical error handler.  1 = Reported directly to the caller by way of return code.  Media I/O errors generated through an IOCTL Category 8 function always get reported directly to the caller by way of return code. The Fail-Errors function applies only to non-IOCTL handle-based file I/O calls.  This bit is not inherited by child processes.										
12	No-Cache/Cache flag:  0 = It is advisable for the disk driver to cache the data in I/O operations on this file.  1 = I/O to the file need not be done through the disk driver cache.  This bit advises FSDs and device drivers whether it is worth caching the data. Like the write-through bit, this is a per-handle bit and is not inherited by child processes.										
11	Reserved and must be zero.										
10 – 8	The locality of reference flags contain information about how the application is to access the file. <table><tr><th>Value</th><th>Definition</th></tr><tr><td>000</td><td>No locality known.</td></tr><tr><td>001</td><td>Mainly sequential access.</td></tr><tr><td>010</td><td>Mainly random access.</td></tr><tr><td>011</td><td>Random with some locality.</td></tr></table>	Value	Definition	000	No locality known.	001	Mainly sequential access.	010	Mainly random access.	011	Random with some locality.
Value	Definition										
000	No locality known.										
001	Mainly sequential access.										
010	Mainly random access.										
011	Random with some locality.										
7	Inheritance flag:  0 = File handle is inherited by a spawned process resulting from a DosExecPgm call.  1 = File handle is private to the current process.  This bit is not inherited by child processes.										
6 – 4	Sharing Mode flags. This field defines any restrictions to file access placed by the caller on other processes: <table><tr><th>Value</th><th>Definition</th></tr><tr><td>001</td><td>Deny Read/Write access</td></tr><tr><td>010</td><td>Deny Write access</td></tr><tr><td>011</td><td>Deny Read access</td></tr><tr><td>100</td><td>Deny neither Read or Write access (Deny None).</td></tr></table> Any other value is invalid.	Value	Definition	001	Deny Read/Write access	010	Deny Write access	011	Deny Read access	100	Deny neither Read or Write access (Deny None).
Value	Definition										
001	Deny Read/Write access										
010	Deny Write access										
011	Deny Read access										
100	Deny neither Read or Write access (Deny None).										

**3** Reserved and must be zero.

**2 – 0** Access Mode flags. This field defines file access required by the caller:

Value	Definition
<b>000</b>	Read – Only access
<b>001</b>	Write – Only access
<b>010</b>	Read/Write access.

Any other value is invalid.

Any other combinations are invalid.

File sharing requires the cooperation of sharing processes. This cooperation is communicated through sharing and access modes. Any sharing restrictions placed on a file opened by a process are removed when the process closes the file with a DosClose request.

### Sharing Mode

Specify the type of access other processes may have to the file (sharing mode). For example, if it is permissible for other processes to continue reading the file while your process is operating on it, specify Deny Write. This sharing mode prevents other processes from writing to the file but still allows them to read it.

### Access Mode

Specify the type of access to the file needed by your process (access mode). For example, if your process requires Read/Write access, and another process has already opened the file with a sharing mode of Deny None, your DosOpen2 request succeeds. However, if the file is open with a sharing mode of Deny Write, your process is denied access.

If the file is inherited by a child process, all sharing and access restrictions are also inherited.

If an open file handle is duplicated by a call to DosDupHandle, all sharing and access restrictions are also duplicated.

### EABuf (PEAOP) – input/output

Address of the extended attribute buffer, which contains an EAOP structure. An EAOP structure has the following format:

#### fpGEAList (PGEALIST)

Address of GEAList. GEAList is a packed array of variable length "get EA" structures, each containing an EA name and the length of the name.

#### fpFEAList (PFEALIST)

Address of FEAList. FEAList is a packed array of variable length "full EA" structures, each containing an EA name and its corresponding value, as well as the lengths of the name and the value.

#### oError (ULONG)

Offset into structure where error has occurred.

On input, the fpGEAList field and oError fields are ignored. The EA setting operation is performed on the information contained in FEAList. If extended attributes are not to be defined or modified, then EABuf must be set to null. Following is the FEAList format:

#### cbList (ULONG)

Length of the FEA list, including the length itself.

#### lList (FEA)

List of FEA structures. An FEA structure has the following format:

#### Flags (BYTE)

Bit indicator describing the characteristics of the EA being defined.

Bit	Description
<b>15</b>	Critical EA.
<b>14 – 0</b>	Reserved and must be set to zero.



If bit 15 is set to 1, this indicates a critical EA. If bit 15 is 0, this means the EA is noncritical; that is, the EA is not essential to the intended use by an application of the file with which it is associated.

**cbName (BYTE)**

Length of EA ASCIIZ name, which does not include the null character.

**cbValue (USHORT)**

Length of EA value, which cannot exceed 64KB.

**szName (PSZ)**

Address of the ASCIIZ name of EA.

**aValue (PSZ)**

Address of the free-format value of EA.

**Note:** The szName and aValue fields are not included as part of header or include files. Because of their variable lengths, these entries must be built manually.

On output, fpGEAList is unchanged. fpFEAList is unchanged as is the area pointed to by fpFEAList. If an error occurred during the set, oError is the offset of the FEA where the error occurred. The API return code is the error code corresponding to the condition generating the error. If no error occurred, oError is undefined.

If EABuf is 0x00000000, then no extended attributes are defined for the file.

**Reserved (ULONG) — input**

Reserved and must be set to zero.

**rc (USHORT) — return**

Return code descriptions are:

0	NO_ERROR
2	ERROR_FILE_NOT_FOUND
3	ERROR_PATH_NOT_FOUND
4	ERROR_TOO_MANY_OPEN_FILES
5	ERROR_ACCESS_DENIED
12	ERROR_INVALID_ACCESS
26	ERROR_NOT_DOS_DISK
32	ERROR_SHARING_VIOLATION
36	ERROR_SHARING_BUFFER_EXCEEDED
82	ERROR_CANNOT_MAKE
87	ERROR_INVALID_PARAMETER
108	ERROR_DRIVE_LOCKED
110	ERROR_OPEN_FAILED
112	ERROR_DISK_FULL
206	ERROR_FILENAME_EXCED_RANGE
231	ERROR_PIPE_BUSY
99	ERROR_DEVICE_IN_USE

### Remarks

A successful DosOpen2 request for a file returns a handle to access the file. The read/write pointer is set at the first byte of the file. The pointer's position may be changed by a DosChgFilePtr request or by read and write operations on the file.

The file's date and time can be queried by calling DosQFileInfo, and is set by calling DosSetFileInfo.

FileAttribute sets attribute bits for the file object. Attributes of an existing file can be queried and set by DosQFileMode and DosSetFileMode. A file's read-only attribute may also be set with the OS/2 ATTRIB command.

FileAttribute cannot be set to Volume Label. Volume labels cannot be opened. DosSetFSInfo may be issued with a logical drive number to set volume label information.

The `FileSize` parameter affects the size of the file only when the file is a new file or a replacement for an existing one. If an existing file is simply opened, `FileSize` is ignored. `DosNewSize` may be called to change the existing file's size.

The value in `FileSize` is a recommended size for the file. If allocation of the full size fails, the open may still succeed. The file system makes a reasonable attempt to allocate the new size in as nearly contiguous an area as possible on the medium. When the file size is extended, the value of the new bytes is undefined.

The DASD Open bit provides direct access to an entire disk or diskette volume, independent of the file system. This mode of opening the volume currently mounted on the drive returns a handle to the caller, which represents the logical volume as a single file. The caller specifies this handle with a `DosDevIOCtl` Category 8 Function 0 request to block other processes from accessing the logical volume.

The file handle state bits can be set by the `DosOpen2` and `DosSetFHandState` requests. An application can query the file handle state bits as well as the rest of the Open Mode field, by calling `DosQFHandState`.

Extended attributes can be set by an EAOP structure in `EABuf` when creating a file, replacing an existing file, or truncating an existing file. No extended attributes are set when simply opening an existing file.

A replace operation is logically equivalent to atomically deleting and re-creating the file. This means that any extended attributes associated with the file are also deleted before the file is re-created.

### **Family API Considerations**

Some options operate differently in the DOS mode than in the OS/2 mode. Therefore, the following restrictions apply to `DosOpen` when coding for the DOS mode:

- **OpenMode restrictions:**
  - The high word of `OpenMode` is ignored and is not error checked.
  - Handles returned in response to DASD open are valid only for `DosDevIOCtl`.
  - Inheritance flag is not supported in DOS 2.X.
  - Write-Through flag must be set to zero.
  - Fail-errors flag must be set to zero.
  - Sharing mode field has meaning only if Sharing is loaded in DOS 3.X, ignored if Sharing is not loaded. Sharing mode is not supported in DOS 2.X.
  - Access mode field has meaning only if Sharing is loaded in DOS 3.X, ignored if Sharing is not loaded. Access mode field is not supported in DOS 2.X.
- Access mode is valid only if Sharing is loaded.

### C Language

```
typedef struct _GEA {      /* gea */

    BYTE cbName;           /* name length not including NULL */
    CHAR szName[1];        /* attribute name */

} GEA;

typedef struct _GEALIST {  /* geal */

    ULONG cbList;          /* total bytes of structure including full list */
    GEA list[1];           /* variable length GEA structures */

} GEALIST;

typedef struct _FEA {      /* fea */

    BYTE fEA;              /* flags */
    BYTE cbName;           /* name length not including NULL */
    USHORT cbValue;        /* value length */

} FEA;

typedef struct _FEALIST {  /* feal */

    ULONG cbList;          /* total bytes of structure including full list */
    FEA list[1];           /* variable length FEA structures */

} FEALIST;

typedef struct _EAOP {     /* eaop */

    PGEALIST fpGEAList;    /* general EA list */
    PFEALIST fpFEAList;    /* full EA list */
    ULONG oError;

} EAOP;

#define INCL_DOSFILEMGR

USHORT rc = DosOpen2(FileName, FileHandle, ActionTaken, FileSize,
                    FileAttribute, OpenFlag, OpenMode, EABuf, Reserved);

PSZ      FileName;        /* File path name string */
PHFILE   FileHandle;       /* File handle (returned) */
PUSHORT  ActionTaken;      /* Action taken (returned) */
ULONG    FileSize;        /* File primary allocation */
USHORT   FileAttribute;    /* File Attribute */
USHORT   OpenFlag;        /* Open function type */
ULONG    OpenMode;        /* Open mode of the file */
PEAOP    EABuf;           /* Extended attribute buffer */
ULONG    0;               /* Reserved (must be zero) */

USHORT   rc;              /* return code */
```

**Assembler Language**

```

GEA    struc

    gea_cbName    db    ?            ;name length not including NULL
    gea_szName    db    1 dup (?)    ;attribute name

GEA    ends

GEALIST    struc

    geal_cbList    dd    ?            ;total bytes of structure including full list
    geal_list      db    size GEA * 1 dup (?) ;variable length GEA structures

GEALIST    ends

FEA    struc

    fea_fEA        db    ? ;flags
    fea_cbName      db    ? ;name length not including NULL
    fea_cbValue     dw    ? ;value length

FEA    ends

FEALIST    struc

    feal_cbList     dd    ?            ;total bytes of structure including full list
    feal_list       db    size FEA * 1 dup (?) ;variable length FEA structures

FEALIST    ends

EAOP    struc

    eaop_fpGEAList  dd    ? ;general EA list
    eaop_fpFEAList  dd    ? ;full EA list
    eaop_oError     dd    ? ;

EAOP    ends

EXTRN  DosOpen2:FAR
INCL_DOSFILEMGR    EQU 1

PUSH@  ASCIIZ  FileName      ;File path name string
PUSH@  WORD    FileHandle     ;File handle (returned)
PUSH@  WORD    ActionTaken    ;Action taken (returned)
PUSH   DWORD   FileSize       ;File primary allocation
PUSH   WORD    FileAttribute   ;File Attribute
PUSH   WORD    OpenFlag       ;Open function type
PUSH   DWORD   OpenMode       ;Open mode of the file
PUSH@  OTHER   EABuf          ;Extended attribute buffer
PUSH   DWORD   0              ;Reserved (must be zero)
CALL   DosOpen2

Returns WORD

```

# DosOpenQueue — Open Queue

---

This call opens a queue for the current process.

<b>DosOpenQueue</b> (OwnerPID, QueueHandle, QueueName)
--

## Parameters

**OwnerPID** (*PUSHORT*) — output

Address of the process ID of the queue owner.

**QueueHandle** (*PHQUEUE*) — output

Address of the write handle of the queue.

**QueueName** (*PSZ*) — input

Address of the name of the queue provided by a previous `DosCreateQueue` call.

**rc** (*USHORT*) — return

Return code descriptions are:

0	NO_ERROR
334	ERROR_QUE_NO_MEMORY
343	ERROR_QUE_NAME_NOT_EXIST

## Remarks

A process that creates a queue has access to it with the handle returned by `DosCreateQueue`. Before another process can place elements in the queue with `DosWriteQueue`, the process must first obtain the queue handle by issuing `DosOpenQueue`.

When specifying the name for the queue, the ASCIIZ name string provided must include the prefix `\QUEUES\`.

## C Language

```
#define INCL_DOSQUEUES
```

```
USHORT rc = DosOpenQueue(OwnerPID, QueueHandle, QueueName);
```

```
PUSHORT OwnerPID; /* Address to put queue owners' PID */
PHQUEUE QueueHandle; /* Address to put handle of queue */
PSZ QueueName; /* Pointer to queue name string */
```

```
USHORT rc; /* return code */
```

## Assembler Language

```
EXTRN DosOpenQueue:FAR
INCL_DOSQUEUES EQU 1
```

```
PUSH@ WORD OwnerPID ;Queue owners' PID (returned)
PUSH@ WORD QueueHandle ;Queue handle (returned)
PUSH@ ASCIIZ QueueName ;Queue name string
CALL DosOpenQueue
```

Returns WORD

# DosOpenSem – Open Existing System Semaphore

This call opens an existing system semaphore that has been created by another process.

**DosOpenSem (SemHandle, SemName)**

## Parameters

**SemHandle (PHSEM)** – output

Address of the handle of the system semaphore created by DosCreateSem. This handle must be supplied by any requests directed to the same semaphore.

**SemName (PSZ)** – input

Address of the name of the system semaphore created by using DosCreateSem.

**rc (USHORT)** – return

Return code descriptions are:

<b>0</b>	<b>NO_ERROR</b>
<b>100</b>	<b>ERROR_TOO_MANY_SEMAPHORES</b>
<b>123</b>	<b>ERROR_INVALID_NAME</b>
<b>187</b>	<b>ERROR_SEM_NOT_FOUND</b>

## Remarks

A system semaphore that is created by a call to DosCreateSem with NoExclusive = 1 specified returns a handle, which the creating process uses to access the semaphore. Another process that needs to access the semaphore must call DosOpenSem after the semaphore's creation. DosOpenSem merely returns the handle of the semaphore; it does not test or change the value of the semaphore.

If a process has access to any semaphores at the time it issues DosExecPgm to start a child process, the new process inherits the handles to these semaphores. However, the new process initially does not own any inherited semaphores, even if the parent process owns them at the time the child process is started. Semaphore ownership is obtained by a successful DosSemRequest call, and only one process can own the semaphore at a time.

When the last process that opened a particular semaphore with a DosCreateSem or DosOpenSem request closes its handle with a DosCloseSem request, the semaphore is deleted from memory and must be redefined by its next user with a call to DosCreateSem.

## C Language

```
#define INCL_DOSSEMAPHORES

USHORT rc = DosOpenSem(SemHandle, SemName);

PHSEM      SemHandle;    /* Semaphore handle (returned) */
PSZ        SemName;      /* Semaphore name string */

USHORT      rc;           /* return code */
```

## Assembler Language

```
EXTRN DosOpenSem:FAR
INCL_DOSSEMAPHORES EQU 1

PUSH@ DWORD SemHandle ;Semaphore handle (returned)
PUSH@ ASCIIZ SemName ;Semaphore name string
CALL DosOpenSem
```

Returns WORD

# DosOpenSem —

## Open Existing System Semaphore

### Example

The following example illustrates the notification of an event between threads of different processes. Process1 creates a nonexclusive system semaphore named process1.sem and sets it. It then invokes process2 to run asynchronously. Process1 then waits, with a timeout of 4.5 seconds, for process2 to open the semaphore and clear it, thereby notifying process1 to resume. Notice that there is a small possibility of process1 missing the notification because process2 may clear the semaphore before process1 issues DosSemWait. See the example for DosSemSetWait for an alternative that would correct this.

```
/* ----- process1.c ---- */

#define INCL_DOSSEMAPHORES
#define INCL_DOSPROCESS

#include <os2.h>

#define SEM_NAME "\\SEM\\process1.sem" /* Semaphore name */
#define TIMEOUT 4500L /* Timeout period */
#define START_PROGRAM "process2.exe" /* Name of program file */

main()
{
    HSEM          SemHandle;
    CHAR          ObjFail [50];
    PSZ           Args;
    PSZ           Envs;
    RESULTCODES   ReturnCodes;
    USHORT        rc;

    printf("Process1 now running. \n");
    rc = DosCreateSem(CSEM_PUBLIC, /* Ownership - nonexclusive */
                    &SemHandle, /* Semaphore handle (returned) */
                    SEM_NAME); /* Semaphore name string */
    printf("Process1.sem created; return code is %d \n", rc);

    /*** SET the semaphore. ***/
    if((rc = DosSemSet(SemHandle))) /* Semaphore handle */

    /***/
    {
        /* Cannot set semaphore -- error processing */
    }
    /* Start a separate process */
    if(!DosExecPgm(ObjFail, /* Object name buffer */
                  sizeof(ObjFail), /* Length of obj. name buffer */
                  EXEC_ASYNC, /* Execution flag - asynchronous */
                  Args, /* Ptr. to argument string */
                  Envs, /* Ptr. to environment string */
                  &ReturnCodes, /* Ptr. to resultcodes struct. */
                  START_PROGRAM)) /* Name of program file */
        printf("Process2 started. Process1 will try to wait for notification. \n");

    /*** WAIT for a notification from process2 on process1.sem ***/
    if((rc = DosSemWait(SemHandle, /* Semaphore handle */
                       TIMEOUT))) /* Timeout period */

    /***/
    {
        /* No notification (either interrupt or timeout); error processing */
    }
    else
    {
        /* Notification received. Normal processing */
        printf("Process2 cleared semaphore; process1 running again. \n");
    }
}
```

## DosOpenSem — Open Existing System Semaphore

```
/* ----- process2.c -----*/

#define INCL_DOSSEMAPHORES

#include <os2.h>

#define SEM_NAME "\\SEM\\process1.sem" /* Semaphore name */

main()
{
    HSEM SemHandle;
    USHORT rc;

    /* Obtain access to semaphore created by process1 via OPEN */
    if((rc=DosOpenSem(&SemHandle, /* Semaphore handle (returned) */
                     SEM_NAME)) /* Semaphore Name */
        /* ***** */
    {
        /* Could not open -- error processing (switch on rc). */
    }
    else
    {
        /* Opened semaphore; normal processing. Clear semaphore when done. */
        printf("Semaphore OPENED. \n");
        if(!rc=DosSemClear(SemHandle)) /* Semaphore handle */
            printf("Semaphore CLEARED. \n");
    }
}
```



# DosPeekNmPipe — Peek Named Pipe

---

This call reads pipe without removing the read data from the pipe.

<b>DosPeekNmPipe</b> ( <i>Handle</i> , <i>Buffer</i> , <i>BufferLen</i> , <i>BytesRead</i> , <i>BytesAvail</i> , <i>PipeState</i> )
---

## Parameters

**Handle** (*HPIPE*) — input

Handle of the named pipe, returned by `DosMakeNmPipe` or `DosOpen`.

**Buffer** (*PBYTE*) — output

Address of the output buffer.

**BufferLen** (*USHORT*) — input

Number of bytes to be read.

**BytesRead** (*PUSHORT*) — output

Address of the number of bytes read.

**BytesAvail** (*PUSHORT*) — output

Address of the 4-byte buffer where the system returns the number of bytes that were available. The buffer structure is:

**2 Bytes** Bytes buffered in pipe (including message header bytes and bytes peeked).

**2 Bytes** Bytes in current message (zero for a byte stream pipe).

**PipeState** (*PUSHORT*) — output

Address of a value representing the state of the named pipe.

Value	Definition
1	Disconnected
2	Listening
3	Connected
4	Closing

The pipe is in a **disconnected** state immediately after:

- A `DosMakeNmPipe` but before the first `DosConnectNmPipe`, or
- A `DosDisconnectNmPipe` but before the next `DosConnectNmPipe`.

A disconnected pipe has no client end and cannot accept a `DosOpen`. The serving end must issue `DosConnectNmPipe`, so an open can be accepted.

The pipe is in a **listening** state after a `DosConnectNmPipe`. A listening pipe is ready to accept a `DosOpen` request. If the pipe is not in a listening state, `DosOpen` returns `ERROR_PIPE_BUSY`.

The pipe is in a **connected** state after a successful `DosOpen` to the listening pipe. The connected pipe allows the serving and client ends to perform I/O, provided both have valid handles.

The pipe is in a **closing** state after the last `DosClose` to the pipe from either the client or server end.

After `DosClose` has been issued for the client handle and any duplicates, and the serving end has read all buffered data, the serving end is returned `ERROR_EOF` for reads and `ERROR_BROKEN_PIPE` for writes. The serving end must acknowledge the closing of the client end by issuing either `DosDisconnectNmPipe` or `DosClose`. Issuing `DosClose` deallocates the pipe.

**rc** (*USHORT*) — return

Return code descriptions are:

0	NO_ERROR
230	ERROR_BAD_PIPE
231	ERROR_PIPE_BUSY
233	ERROR_PIPE_NOT_CONNECTED

# DosPeekNmPipe — Peek Named Pipe

## Remarks

DosPeekNmPipe is similar to a DosRead, except the bytes read from the pipe are not removed. In addition, a DosPeekNmPipe never blocks, regardless of the blocking mode set. If the pipe cannot be accessed immediately, ERROR\_PIPE\_BUSY is returned. Because the peek cannot block, it returns only what is currently in the pipe. Thus, a portion of a message may be returned, even though the size of the peek can accommodate the entire message.

The value returned in PipeState can be used by the client or the server end to determine the current state of the pipe and take appropriate action.

## C Language

```
#define INCL_DOSNMPPIPES

USHORT rc = DosPeekNmPipe(Handle, Buffer, BufferLen, BytesRead,
                          BytesAvail, PipeState);

HPIPE      Handle;      /* Pipe handle */
PBYTE      Buffer;       /* Address of user buffer */
USHORT     BufferLen;    /* Buffer length */
PUSHORT    BytesRead;   /* Bytes read (returned) */
PUSHORT    BytesAvail;  /* Bytes available (returned) */
PUSHORT    PipeState;   /* Pipe state (returned) */

USHORT     rc;          /* return code */
```

## Assembler Language

```
EXTRN DosPeekNmPipe:FAR
INCL_DOSNMPPIPES EQU 1

PUSH WORD Handle ;Pipe handle
PUSH@ OTHER Buffer ;User buffer
PUSH WORD BufferLen ;Buffer length
PUSH@ WORD BytesRead ;Bytes read (returned)
PUSH@ DWORD BytesAvail ;Bytes available (returned)
PUSH@ WORD PipeState ;Pipe state (returned)
CALL DosPeekNmPipe

Returns WORD
```

# DosPeekQueue — Peek Queue

---

This call retrieves an element from a queue without removing it from the queue.

**DosPeekQueue** (*QueueHandle*, *Request*, *DataLength*, *DataAddress*, *ElementCode*,  
*NoWait*, *ElemPriority*, *SemaphoreHandle*)

## Parameters

**QueueHandle** (*HQUEUE*) — input  
Handle of the queue to read.

**Request** (*PULONG*) — output  
Address of the data to be filled in with the following information.

Word	Description
1	The PID of the process that added the element to the queue.
2	Used for event coding by the application. The data in this word is the same as that furnished by the Request parameter on the DosWriteQueue request for the corresponding queue element. The value of this data is understood by the client thread and by the server thread. There is no special meaning to this data and the operating system does not alter the data.

**DataLength** (*PUSHORT*) — output  
Address of the length of the received data.

**DataAddress** (*PULONG*) — output  
Address of the element retrieved from the queue.

**ElementCode** (*PUSHORT*) — input/output  
Address of the code that identifies the element to be read. This field is set to:

Value	Definition
0	Read the element at the beginning of the queue and return the identifier of the next element.
≠0	Read the element whose identifier is specified and return the identifier of the next element.

**NoWait** (*UCHAR*) — input  
Action to be performed when there are no entries on the queue, shown below.

Value	Definition
0	Requesting thread waits
1	Requesting thread does not wait.

**ElemPriority** (*PBYTE*) — output  
Address of the element's priority. This is the value that was specified for ElemPriority by the DosWriteQueue call that added the element to the queue.

**SemaphoreHandle** (*ULONG*) — input  
Handle of a semaphore to be cleared when the queue has data placed into it and NoWait=0 is specified. If NoWait=1 is specified, this parameter should be set to null.

The semaphore may be either a RAM or system semaphore. If this handle is for a RAM semaphore, that semaphore must be in a segment shared between the process that owns the queue and any process that issues a DosWriteQueue request to the queue.

If multiple threads are processing elements from the queue using a NoWait=0, the same semaphore must be provided on all DosPeekQueue or DosReadQueue requests.

**rc** (*USHORT*) — return  
Return code descriptions are:

## DosPeekQueue – Peek Queue

0	NO_ERROR
330	ERROR_QUE_PROC_NOT_OWNED
333	ERROR_QUE_ELEMENT_NOT_EXIST
337	ERROR_QUE_INVALID_HANDLE
342	ERROR_QUE_EMPTY
433	ERROR_QUE_INVALID_WAIT

### Remarks

A process that creates a queue with `DosCreateQueue` owns it. Only the owning process and any threads it creates can issue `DosPeekQueue` to examine a queue element without removing it from the queue. If the queue is empty and `NoWait=0` is specified, the thread waits for an element to be added to the queue. If the queue is empty and `NoWait=1` is specified, the thread returns with `ERROR_QUE_EMPTY`.

The `ElementCode` parameter returns an indicator for the element being examined. To examine the first element in the queue, the queue owner issues `DosPeekQueue` with `ElementCode` set to zero. To examine the next element in the queue, the queue owner uses the value returned in `ElementCode` as input for the next `DosPeekQueue` request, and so on. By issuing successive peeks, the queue owner can select a queue element and then remove it from the queue by specifying an `ElementCode` value with a `DosReadQueue` request.

The semaphore provided by `SemaphoreHandle` is typically used with a `DosMuxSemWait` request to wait for a queue or any of several other events. If `DosReadQueue` is issued with `NoWait=0`, it clears the semaphore indicated by `SemaphoreHandle` as soon as the element is peeked.

The `Request`, `DataLength` and `DataBuffer` parameters contain data understood by the thread adding the element to the queue and by the thread that receives the queue element. There is no special meaning to this data; applications may use these parameters for any purpose they wish. OS/2 does not alter this data; it simply copies this data intact. OS/2 does not validate the address of `DataBuffer` or the `DataLength`.

### C Language

```
#define INCL_DOSQUEUES
```

```
USHORT rc = DosPeekQueue(QueueHandle, Request, DataLength, DataAddress,  
                          ElementCode, NoWait, ElemPriority, SemaphoreHandle);
```

```
HQUEUE      QueueHandle;    /* Queue handle */  
PULONG      Request;        /* Request identification data (returned) */  
PUSHORT     DataLength;     /* Length of element received (returned) */  
PULONG      DataAddress;    /* Address of element received (returned) */  
PUSHORT     ElementCode;    /* Indicator of element received (returned) */  
UCHAR       NoWait;         /* Indicate to not wait if queue is empty */  
PBYTE       ElemPriority;    /* Priority of element (returned) */  
ULONG       SemaphoreHandle; /* Semaphore Handle */
```

```
USHORT      rc;              /* return code */
```

### Assembler Language

```
EXTRN DosPeekQueue:FAR  
INCL_DOSQUEUES EQU 1
```

```
PUSH WORD QueueHandle ;Queue handle  
PUSH@ DWORD Request ;Request identification data (returned)  
PUSH@ WORD DataLength ;Length of element received (returned)  
PUSH@ DWORD DataAddress ;Address of element received (returned)  
PUSH@ WORD ElementCode ;Indicator of element received (returned)  
PUSH OTHER NoWait ;Indicate to not wait if queue is empty  
PUSH@ BYTE ElemPriority ;Priority of element (returned)  
PUSH DWORD SemaphoreHandle ;Semaphore Handle  
CALL DosPeekQueue
```

Returns WORD

# DosPFSActivate —

## Activate Font

---

This call specifies the code page and font to make active for the specified printer and system file number.

<b>DosPFSActivate</b> ( <b>SpiHandle</b> , <b>BytesWritten</b> , <b>PrinterName</b> , <b>CodePage</b> , <b>FontID</b> , <b>SFN</b> , <b>Reserved</b> )
--

### Parameters

**SpiHandle** (*PVOID*) — input

Address of the file handle of the temporary spool file that activates code page and font switching.

**BytesWritten** (*PULONG*) — output

Address of the number of bytes written to the temporary spool file.

**PrinterName** (*PSZ*) — input

Address of the name of the printer that activates code page and font switching.

**CodePage** (*USHORT*) — input

Active code page for the specified printer and system file number.

**FontID** (*USHORT*) — input

Active font within the specified code page for the specified printer and system file number.

For download fonts, the FontID is that specified in the printer font file.

For cartridge fonts, the FontID is the number specified on the label of the cartridge and in the DEVINFO statement for the printer.

A value of 0 (0000H) for both the CodePage and FontID indicates that the hardware default code page and font should be made active.

A value of 0 for the font ID but not the code page indicates that any font ID is acceptable for the code pages.

**SFN** (*USHORT*) — input

System file number of the requester. The SFN is passed as a parameter in the monitor packet.

**Reserved** (*ULONG*) — input

Reserved must be set to zero.

**rc** (*USHORT*) — return

Return code descriptions are listed in following section.

### Remarks

DosPFSActivate is intended for use only by applications that replace the spooler as a print monitor and that do code page switching. Other applications should use printer IOCTLs to manipulate printer code page switching.

DosPFSActivate is located in SPOOLCP.DLL (not in DOSCALLS.LIB) and requires an import statement in the module definition file. See the *IBM Operating System/2 Version 1.2 Building Programs*, Module Definition File Statements section for information regarding the import statement.

Return values are:

2	Code page not available.
4	Font ID not available.
9	Code page switcher internal error.
10	Invalid printer name as input.
13	Received code page request when code page switcher not initialized.
15	System file number table full. Cannot activate another entry.
19	I/O error reading font file control sequence section.
21	I/O error reading font file font definition block.
23	I/O error while writing to temporary spool file.

## DosPFSActivate — Activate Font

**24**           Disk full error while writing to temporary spool file.  
**25**           Bad spool file handle.

### C Language

```
#define INCL_DOSPFS

USHORT rc = DosPFSActivate(SplHandle, BytesWritten, PrinterName,
                           CodePage, FontID, SFN, Reserved);

PVOID      SplHandle;      /* Temporary Spool File handle */
PULONG     BytesWritten;   /* Number of bytes written (returned) */
PSZ        PrinterName;   /* Printer name string */
USHORT     CodePage;       /* Code Page to make active */
USHORT     FontID;         /* Font ID to make active */
USHORT     SFN;            /* System File Number */
ULONG      0;              /* Reserved, set to zero */

USHORT     rc;              /* return code */
```

### Assembler Language

```
EXTRN DosPFSActivate:FAR
INCL_DOSPFS EQU 1

PUSH@ DWORD SplHandle      ;Temporary Spool File handle
PUSH@ DWORD BytesWritten   ;Number of bytes written (returned)
PUSH@ ASCIIZ PrinterName   ;Printer name string
PUSH WORD CodePage         ;Code Page to make active
PUSH WORD FontID           ;Font ID to make active
PUSH WORD SFN              ;System File Number
PUSH DWORD 0               ;Reserved (must be zero)
CALL DosPFSActivate
```

Returns WORD

# DosPFSCloseUser – Close Font User Interface

This call indicates to the Font Switcher that the specified process has closed its spool file. The Font Switcher may then free any resources being used to track code page and font switching for a process.

**DosPFSCloseUser (PrinterName, SFN, Reserved)**

## Parameters

**PrinterName (PSZ)** – input

Address of the name of the printer that closes code page and font switching.

**SFN (USHORT)** – input

System File Number of the requester. The SFN is passed as a parameter in the monitor packet.

**Reserved (ULONG)** – input

Reserved must be set to zero.

**rc (USHORT)** – return

Return code descriptions are listed in the following section.

## Remarks

DosPFSCloseUser is intended for use only by applications that replace the spooler as a print monitor and that do code page switching. Other applications should use printer IOCTLs to manipulate printer code page switching.

DosPFSCloseUser is located in SPOOLCP.DLL (not in DOSCALLS.LIB) and requires an import statement in the module definition file. Refer to the *IBM Operating System/2 Version 1.2 Building Programs*, Module Definition File Statements section for information regarding the import statement.

Return values are:

- 8**            Attempted to close system file number not active.
- 9**            Code page switcher internal error.
- 10**          Invalid printer name as input.
- 13**          Received code page request when code page switcher not initialized.

## C Language

```
#define INCL_DOSPFPS
```

```
USHORT rc = DosPFSCloseUser(PrinterName, SFN, Reserved);
```

```
PSZ      PrinterName; /* Printer name string */
USHORT   SFN;         /* System File Number */
ULONG    0;           /* Reserved, set to zero */

USHORT   rc;          /* return code */
```

## Assembler Language

```
EXTRN DosPFSCloseUser:FAR
INCL_DOSPFPS EQU 1
```

```
PUSH@ ASCIIZ PrinterName ;Printer name string
PUSH WORD SFN ;System File Number
PUSH DWORD 0 ;Reserved (must be zero)
CALL DosPFSCloseUser
```

Returns WORD

## DosPFSInit – Initialize Code Page and Font

This call allows the Font Switcher to initialize code page and font switching for a specified printer.

<b>DosPFSInit (CPHdw, FontFileName, PrinterType, PrinterName, Instances, Reserved)</b>
--

### Parameters

**CPHdw (USHORT)** – input

Address of the pointer list, in the following format, that specifies the hardware code page and fonts on the printer.

Word	Description
------	-------------

<b>Word 0</b>	Number of definitions that follow.
---------------	------------------------------------

<b>DWord 1 / N</b>	Code page number (1st Word of DWord) and Font ID (2nd Word of DWord) for each hardware font in order corresponding to the hardware code page and font selection numbers. (For example, the first code page and font ID value corresponds to the default hardware font 0, the second, to hardware font 1, the third, to hardware font 2, and so on. If the default hardware font is not known, 0 should be specified for the default code page and font).
--------------------	--

**FontFileName (PSZ)** – input

Address of the pathname of the font file of the specified printer that initiates the code page and font switching.

**PrinterType (PSZ)** – input

Address of the printer type ID.

**PrinterName (PSZ)** – input

Address of the name of the printer that initiates code page and font switching.

**Instances (USHORT)** – input

Maximum number of different instances of use tracking code page and font switching. This value is advisory for the Font Switcher allocating enough resources for the specified number of instances being tracked.

**Reserved (ULONG)** – input

Reserved must be set to zero.

**rc (USHORT)** – return

Return code descriptions are listed in the following section.

### Remarks

DosPFSInit is intended for use only by applications that replace the spooler as a print monitor and that do code page switching. Other applications should use printer IOCTLs to manipulate printer code page switching.

DosPFSInit is located in SPOOLCP.DLL (not DOSCALLS.LIB) and requires an import statement in the module definition file. Refer to the *IBM Operating System/2 Version 1.2 Building Programs*, Module Definition File Statements section for information regarding the import statement.

Return values are:

- |    |   |
|----|---|
| 1  | Code page switcher already initialized  |
| 3  | User entered too many ROMs in DEVINFO, initialization continued with the rest |
| 6  | Wrong or missing font file ID   |
| 9  | Code page switcher internal error   |
| 10 | Invalid printer name as input   |
| 11 | Printer type input does not match that in font file                           |
| 12 | Could not get storage for control blocks                                      |
| 14 | Could not open font file during initialization                                |



## DosPFSInit — Initialize Code Page and Font

**17** Switcher reports too many system file number entries  
**19** I/O error reading font file control sequence section  
**20** I/O error reading font file header  
**21** I/O error reading font file font definition block  
**22** Some fonts bad due to error in font file, initialization continued.

### C Language

```
#define INCL_DOSPFS

USHORT rc = DosPFSInit(CPHdw, FontFileName, PrinterType, PrinterName,
                      Instances, Reserved);

PUSHORT    CPHdw;        /* Hdw Font definition list */
PSZ        FontFileName; /* File pathname of the font file to use */
PSZ        PrinterType;  /* Printer type string */
PSZ        PrinterName;  /* Printer name string */
USHORT     Instances;    /* Number of spool instances */
ULONG      0;            /* Reserved */

USHORT     rc;            /* return code */
```

### Assembler Language

```
EXTRN DosPFSInit:FAR
INCL_DOSPFS EQU 1

PUSH@ WORD CPHdw ;Hdw Font definition list
PUSH@ ASCIIZ FontFileName ;File pathname of the font file to use
PUSH@ ASCIIZ PrinterType ;Printer type string
PUSH@ ASCIIZ PrinterName ;Printer name string
PUSH WORD Instances ;Number of spool instances
PUSH DWORD 0 ;Reserved (must be zero)
CALL DosPFSInit
```

Returns WORD

# DosPFSQueryAct – Query Active Font

This call queries the active code page and font for the specified printer and system file number.

**DosPFSQueryAct (PrinterName, CodePage, FontID, SFN, Reserved)**

## Parameters

**PrinterName (PSZ)** – input

Address of the name of the printer that queries the active code page and font.

**CodePage (USHORT)** – output

Address of the active code page that returns the specified printer and System File Number.

**FontID (USHORT)** – output

Address of the active Font ID number that returns the specified printer and System File Number.

**SFN (USHORT)** – input

System File Number of the requester. The SFN is passed as a parameter in the monitor packet.

**Reserved (ULONG)** – input

Reserved must be set to zero.

**rc (USHORT)** – return

Return code descriptions are listed in the following section.

## Remarks

DosPFSQueryAct is intended for use only by applications that replace the spooler as a print monitor and that do code page switching. Other applications should use printer IOCTLs to manipulate printer code page switching.

DosPFSQueryAct is located in SPOOLCP.DLL (not in DOSCALLS.LIB) and requires an import statement in the module definition file. Refer to the *IBM Operating System/2 Version 1.2 Building Programs*, Module Definition File Statements section for information regarding the import statement.

Return values are:

- |           |  |
|-----------|--|
| <b>9</b>  | Code page switcher internal error.   |
| <b>10</b> | Invalid printer name as input.   |
| <b>13</b> | Received code page request when code page switcher not initialized.          |
| <b>16</b> | Received request for system file number not in the system file number table. |

## C Language

```
#define INCL_DOSPFS
```

```
USHORT rc = DosPFSQueryAct(PrinterName, CodePage, FontID, SFN, Reserved);
```

```
PSZ      PrinterName; /* Printer name string */
USHORT   CodePage;    /* Code Page return */
USHORT   FontID;      /* Font ID return */
USHORT   SFN;         /* System File Number */
ULONG    0;           /* Reserved, set to zero */

USHORT   rc;          /* return code */
```

# DosPFSQueryAct — Query Active Font

## Assembler Language

```
EXTRN DosPFSQueryAct:FAR
INCL_DOSPFS EQU 1

PUSH@ ASCIIZ PrinterName ;Printer name string
PUSH@ WORD CodePage ;Code Page return
PUSH@ WORD FontID ;Font ID return
PUSH WORD SFN ;System File Number
PUSH DWORD 0 ;Reserved (must be zero)
CALL DosPFSQueryAct
```

Returns WORD

# DosPFSVerifyFont – Verify Font

This call indicates whether the specified code page and font within that code page are available in the font file for the specified printer.

**DosPFSVerifyFont (PrinterName, CodePage, FontID, Reserved)**

## Parameters

**PrinterName (PSZ)** – input

Address of the name of the printer that queries the code page and font switching setup.

**CodePage (USHORT)** – input

Code page to validate. Values are in the range 0 through 65535.

**FontID (USHORT)** – input

Font ID to validate. Values are in the range 0 through 65535. A value of 0 indicates that any font within the specified code page is acceptable.

**Reserved (ULONG)** – input

Reserved must be set to zero.

**rc (USHORT)** – return

Return code descriptions are listed in the following section.

## Remarks

DosPFSVerifyFont is intended for use only by applications that replace the spooler as a print monitor and that do code page switching. Other applications should use printer IOCTLs to manipulate printer code page switching.

DosPFSVerifyFont is located in SPOOLCP.DLL (not in DOSCALLS.LIB) and requires an import statement in the module definition file. Refer to the *IBM Operating System/2 Version 1.2 Building Programs*, Module Definition File Statements section for information about the import statement.

Return values are:

- |           |   |
|-----------|---|
| <b>2</b>  | Code page not available.  |
| <b>4</b>  | Font ID not available.  |
| <b>10</b> | Invalid printer name as input.                                      |
| <b>13</b> | Received code page request when code page switcher not initialized. |

## C Language

```
#define INCL_DOSPFS
```

```
USHORT rc = DosPFSVerifyFont(PrinterName, CodePage, FontID, Reserved);
```

```
PSZ      PrinterName; /* Printer name string */
USHORT   CodePage;    /* Code Page to validate */
USHORT   FontID;      /* Font Id to validate */
ULONG    0;           /* Reserved, set to zero */

USHORT   rc;          /* return code */
```

# DosPFSVerifyFont — Verify Font

## Assembler Language

```
EXTRN DosPFSVerifyFont:FAR
INCL_DOSPFS EQU 1

PUSH@ ASCIIZ PrinterName ;Printer name string
PUSH WORD CodePage ;Code Page to validate
PUSH WORD FontID ;Font Id to validate
PUSH DWORD 0 ;Reserved (must be zero)
CALL DosPFSVerifyFont
```

Returns WORD

# DosPhysicalDisk – Get Information about Partitionable Disk

---

This call obtains information about partitionable disks.

**DosPhysicalDisk (Function, DataPtr, DataLen, ParmPtr, ParmLen)**

## Parameters

**Function** (*USHORT*) – input

The functions supported are:

Value	Definition
1	Obtain total number of partitionable disks
2	Obtain a handle to use with Category 9 IOCTLs
3	Release a handle for a partitionable disk.

**DataPtr** (*PBYTE*) – output

Address of the buffer where the returned information is placed.

**DataLen** (*USHORT*) – input

Length of the data buffer. The output data for each function is described below.

Function	DataLen	Returned Information
1	2	Total number of partitionable disks in the system, 1-based.
2	2	Handle for the specified partitionable disk for the category 9 IOCTLs.
3	0	None. Pointer must be zero.

**Note:** All lengths are in bytes.

**ParmPtr** (*PBYTE*) – input

Address of the buffer used for input parameters.

**ParmLen** (*USHORT*) – input

Length of the parameter buffer. The input parameters required for each function are:

Function	ParmLen	Input Parameters
1	0	None. Must be set to zero.
2	String length	ASCII string that specifies the partitionable disk.
3	2	Handle obtained from function 2.

**Note:** All lengths are in bytes.

**rc** (*USHORT*) – return

Return code descriptions are:

0	NO_ERROR
1	ERROR_INVALID_FUNCTION
5	ERROR_ACCESS_DENIED
6	ERROR_INVALID_HANDLE
33	ERROR_LOCK_VIOLATION
87	ERROR_INVALID_PARAMETER

# DosPhysicalDisk —

## Get Information about Partitionable Disk

### Remarks

The ASCIIZ string used to specify the partitionable disk must be of the following format:

number : <null byte>

Where:

<b>number</b>	specifies the partitionable disk (1-based) number in ASCII
<b>:</b>	must be present
<b>&lt;null byte&gt;</b>	the byte of 0 for the ASCIIZ string

The handle returned for the specified partitionable disk can only be used with the DosDevIOCtl call for the Category 9 Generic IOCTL. Use of the handle for a physical partitionable disk is not permitted for handle-based file system function calls, such as DosRead or DosClose.

### C Language

```
#define INCL_DOSDEVICES

USHORT rc = DosPhysicalDisk(Function, DataPtr, DataLen, ParmPtr, ParmLen);

USHORT      Function;      /* Type of information */
PBYTE       DataPtr;       /* Pointer to return buffer */
USHORT      DataLen;       /* Return buffer length */
PBYTE       ParmPtr;       /* Pointer to user-supplied information */
USHORT      ParmLen;       /* Length of user-supplied information */

USHORT      rc;            /* return code */
```

### Assembler Language

```
EXTRN DosPhysicalDisk:FAR
INCL_DOSDEVICES EQU 1

PUSH WORD Function      ;Type of information
PUSH@ OTHER DataPtr     ;Return buffer (returned)
PUSH WORD DataLen       ;Return buffer length
PUSH@ OTHER ParmPtr     ;User-supplied information
PUSH WORD ParmLen       ;Length of user-supplied information
CALL DosPhysicalDisk

Returns WORD
```

This call requests or releases access to ports for I/O privilege.

**DosPortAccess** (*Reserved*, *TypeOfAccess*, *FirstPort*, *LastPort*)

## Parameters

**Reserved** (*USHORT*) — input  
Must be set to zero.

**TypeOfAccess** (*USHORT*) — input  
A request for or release of access to a port.

Value	Definition
0	Request access
1	Release access.

**FirstPort** (*USHORT*) — input  
Starting (low) number in a contiguous range or a single port.

**LastPort** (*USHORT*) — input  
Ending (high) number in a contiguous range or a single port. If only one port is being used, *FirstPort* and *LastPort* should both be set to this port.

**rc** (*USHORT*) — return  
Return code descriptions are:

0	NO_ERROR
5	ERROR_ACCESS_DENIED

## Remarks

Note that CLI/STI privilege is also granted automatically. There is no need to make an additional call to *DosCLIAccess*.

Applications that perform I/O to port(s) in IOPL segments must request port access from the operating system.

An application with no IOPL segments that accesses a device through a device driver or by an interface package such as VIO, does not need to issue this call. The device driver or interface package is responsible for obtaining the necessary I/O access.

## C Language

```
#define INCL_DOSDEVICES
```

```
USHORT rc = DosPortAccess(Reserved, TypeOfAccess, FirstPort, LastPort);
```

```
USHORT      0;           /* 0 */
USHORT      TypeOfAccess; /* Request or release */
USHORT      FirstPort;    /* First port number */
USHORT      LastPort;     /* Last port number */

USHORT      rc;           /* return code */
```



# DosPortAccess — Request Port Access

FAP1

## Assembler Language

```
EXTRN DosPortAccess:FAR
INCL_DOSDEVICES      EQU 1

PUSH WORD 0           ;Reserved (must be zero)
PUSH WORD TypeOfAccess ;Request or release
PUSH WORD FirstPort   ;First port number
PUSH WORD LastPort    ;Last port number
CALL DosPortAccess
```

Returns WORD

# DosPtrace —

## Provides an Interface for Program Debugging

---

This call provides an interface into the OS/2 kernel to facilitate program debugging.

### DosPtrace (PtraceB)

### Parameters

**PtraceB (PBYTE)** — output

Address of the Ptrace command/data buffer. This buffer is used to communicate between the debug program and the DosPtrace routines.

**rc (USHORT)** — return

Return code descriptions are:

<b>0</b>	<b>NO_ERROR</b>
<b>1</b>	<b>ERROR_INVALID_FUNCTION</b>
<b>5</b>	<b>ERROR_ACCESS_DENIED</b>
<b>115</b>	<b>ERROR_PROTECTION_VIOLATION</b>
<b>303</b>	<b>ERROR_INVALID_PROCID</b>

### Remarks

DosPtrace allows a parent process to control the execution of another process by implementing breakpoint debugging for a debugger. Both the program under test and the program being debugged must be executing in OS/2 mode.

DosPtrace supports debugging of a process with multiple threads by allowing the debugger to read and write registers, freeze and resume thread execution, and get status on the threads.

The debugger must be able to read and write the instructions, data, and registers of the program being debugged to insert breakpoint instructions. When a process runs in OS/2 mode, one process cannot directly manipulate the address space of another process. OS/2 controls this address space through the use of the trace flag facility in DosExecPgm and the Ptrace buffer in DosPtrace.

The steps to program debugging in OS/2 mode follow:

1. The debug program issues DosExecPgm for the program to be debugged, and specifies the trace option.
2. The debug program calls DosPtrace with the TRC\_C\_Stop command to initialize the Ptrace Buffer.
3. The debug program sets up a Ptrace buffer with commands for inserting the breakpoints and issues repeated DosPtrace requests as necessary.
4. The debug program sets up a Ptrace buffer with a command to begin execution and issues DosPtrace. This may be a TRC\_CS\_Step, or TRC\_C\_Go.
5. When the kernel DosPtrace program receives control from the program being debugged, it returns to the debug program with the Ptrace buffer set to the current register contents and with indicators of the reason for return.
6. The kernel DosPtrace program receives control at a breakpoint interrupt, at processor exceptions, or when the program ends.

To debug a process with multiple threads, set a field in the Ptrace buffer (Ptrace\_B.TID) to the thread ID of the thread of interest. This causes the read/write register commands to receive only the register set of the specified thread.

**Note:** For a process with multiple threads, the address space is the same for all the threads in the process. When commands are issued to read/write memory locations or set breakpoints, they affect all the threads in the process, even when a command is issued with a specific thread ID.

The debugger may suspend and resume specific threads through use of the TRC\_C\_Freeze and TRC\_C\_Resume commands. Having only selected threads be affected by the breakpoints is useful for manipulating them while other threads are suspended.

## DosPtrace — Provides an Interface for Program Debugging

When a process debugger terminates, the program being debugged also terminates. To accomplish this, an internal link between the debugger and the program being debugged is maintained. This link is established as a result of the first successful DosPtrace command. Once established, this link can not be reset.

The process being debugged does not need to be a direct child process. In this situation, a small window of time exists between the DosExecPgm call and the first DosPtrace call. If the debugger terminates during this window, the program being debugged cannot be cleaned up. The system automatically terminates the program that was to be debugged.

Specifying a trace option of 2 with DosStartSession enables a debugger running in the parent session to trace all processes associated with an application running in the child session, regardless of whether the process was started by DosStartSession request or by DosExecPgm. See DosStartSession for more information.

### Contents of the Ptrace Buffer:

<i><b>Ptrace_B</b></i>			<i><b>Structure</b></i>
PID	DW	0	; Process ID of the process being debugged
TID	DW	0	; Thread ID of the process being debugged
Cmd	DW	0	; Request to DosPtrace, or DosPtrace result code
Value	DW	?	; Data to DosPtrace, or DosPtrace error code
OffV	DW	?	; Offset value
SegV	DW	?	; Segment value
MTE	DW	?	; Library Module handle
Ptrace B ENDS			

### Exceptions:

<i><b>Ptraceregs</b></i>			<i><b>Structure</b></i>
rAX	DW	?	; Registers AX through SS
rBX	DW	?	
rCX	DW	?	
rDX	DW	?	
rSI	DW	?	
rDI	DW	?	
rBP	DW	?	
rDS	DW	?	
rES	DW	?	
rIP	DW	?	
rCS	DW	?	
rF	DW	?	
rSP	DW	?	
rSS	DW	?	

## DosPtrace –

### Provides an Interface for Program Debugging

<b><i>Ptraceregs</i></b>			<b><i>Structure</i></b>
Ptraceregs ENDS			

For the TRC\_C\_ReadMemB and TRC\_C\_WriteMemB commands, the remainder of the Ptrace buffer contains:

<b><i>Ptraceptr</i></b>			<b><i>Structure</i></b>
OffB	DW	?	; Buffer Address
SegB	DW	?	
Ptraceptr ENDS			

**DosPtrace Commands:** PTrace\_B.Cmd must contain one of the following commands upon entrance to DosPtrace:

TRC_C_Null	EQU 0		; Invalid
TRC_C_ReadMem_I	EQU 1		; Instruction
TRC_C_ReadMem_D	EQU 2		; Data
TRC_C_ReadMem	EQU	TRC_C_ReadMem_I	
TRC_C_ReadReg	EQU 3		
TRC_C_WriteMem_I	EQU 4		; Instruction
TRC_C_WriteMem_D	EQU 5		; Data
TRC_C_WriteMem	EQU	TRC_C_WriteMem_I	
TRC_C_WriteReg	EQU 6		
TRC_C_Go	EQU 7		
TRC_C_Term	EQU 8		
TRC_C_SStep	EQU 9		
TRC_C_Stop	EQU 10		; Initialize
TRC_C_Freeze	EQU 11		
TRC_C_Resume	EQU 12		
TRC_C_NumToSel	EQU 13		
TRC_C_GetFPRegs	EQU 14		
TRC_C_SetFPRegs	EQU 15		
TRC_C_GetLibName	EQU 16		
TRC_C_ThrdStat	EQU 17		
TRC_C_ReadMem_B	EQU 18		; Read Block
TRC_C_WriteMem_B	EQU 19		; Write Block

# DosPtrace —

## Provides an Interface for Program Debugging

**Commands and Required Input:** A command is issued by placing the command number in Ptrace buffer, and calling DosPtrace with that buffer.

All of the commands require that Ptrace\_B.PID be the PID of the process to debug.

**TRC\_C\_Null** Not a valid command

**Memory Operations:** For the following commands, SegV:OffV is the affected location, and Ptrace\_B.Value contains the value to write to or that was read from the debugger's memory. GDT segments cannot be written to: all except privilege level 2 and 3 access is disallowed. A write to a shared code segment makes that segment a non-shared segment before the write.

**TRC\_C\_ReadMem\_I** Read instruction.  
**TRC\_C\_ReadMem\_D** Read data.  
**TRC\_C\_ReadMem** Read any memory.  
**TRC\_C\_WriteMem\_I** Write instruction.  
**TRC\_C\_WriteMem\_D** Write data.  
**TRC\_C\_WriteMem** Write to any memory.

**Block operations:** For the block operations, SegV:OffV must point to the starting address in the debugger's memory, and Value is the number of bytes to copy. On return, Value contains the number of bytes actually copied.

**TRC\_C\_ReadMem\_B** Read memory block.  
**TRC\_C\_WriteMem\_B** Write memory block.

**Register / Thread Operations:** For the following commands, Ptrace\_B.TID must contain the thread ID of the thread in question. If the Ptrace\_B.TID field is zero:

- TRC\_C\_ThrdStat returns the number of threads in the process, (Ptrace\_B.Value).
- TRC\_C\_Freeze and TRC\_C\_Resume affects all of the debugger's threads.

**TRC\_C\_ReadReg** Examine thread's registers.  
**TRC\_C\_WriteReg** Write thread's registers.  
**TRC\_C\_Freeze** Suspend a thread.  
**TRC\_C\_Resume** Resume a suspended thread.  
**TRC\_C\_ThrdStat** Get thread status.

**Command Operations:** For the following commands, the Ptrace\_B.PID must be valid. The Ptrace\_B registers are ignored for these commands. For TRC\_C\_Go and TRC\_C\_SStep, any thread may gain control first. The TRC\_C\_Term command terminates the program being debugged.

**TRC\_C\_Go** Run the program being debugged.  
**TRC\_C\_Term** Terminate the program being debugged.  
**TRC\_C\_SStep** Run one instruction.  
**TRC\_C\_Stop** Initialize PTrace buffer.

**Library Support:** For TRC\_C\_NumToSel, Ptrace\_B.Value should be set to the segment number on entrance, and a valid selector on exit. Also, Ptrace\_B.MTE should be set to the module's handle. The MTE identifies the different library files in the program being debugged.

For TRC\_C\_GetLibName, SegV:OffV should point to a buffer where the name of the library is returned. PTrace\_B.Value should hold the library's module handle (MTE).

**TRC\_C\_NumToSel** Convert Segment number to selector.  
**TRC\_C\_GetLibName** Return name of module.

**Floating Point Support:** For the following two commands, SegV:OffV must contain a pointer to a 94 byte buffer to be used to read/write the floating point registers from/to.

The layout of this area is described in the NPX287 manual under the heading FSAVE/FRSTOR memory layout.

**TRC\_C\_GetFPRegs** Read floating point registers.  
**TRC\_C\_SetFPRegs** Write floating point registers.

## DosPtrace —

# Provides an Interface for Program Debugging

**DosPtrace Return Codes:** When DosPtrace returns to the debug program, the result is placed in Ptrace\_B.Cmd, and reflects the reason for the return.

The values returned are:

TRC_C_SUC_ret	EQU 0	; Success
TRC_C_ERR_ret	EQU -1	; Error
TRC_C_SIG_ret	EQU -2	; Signal
TRC_C_TBT_ret	EQU -3	; Single Step
TRC_C_BPT_ret	EQU -4	; Breakpoint
TRC_C_NMI_ret	EQU -5	; Parity Error
TRC_C_KIL_ret	EQU -6	; Process dying
TRC_C_GPF_ret	EQU -7	; GP fault
TRC_C_LIB_ret	EQU -8	; Library load
TRC_C_FPE_ret	EQU -9	; FP error
TRC_C_THD_ret	EQU -10	; Thread ending
TRC_C_STP_ret	EQU -11	; Async. Stop.

If Ptrace\_B.Cmd is returned as TRC\_C\_ERR\_ret, Ptrace\_B.Value is set to one of the following:

**TRACE\_BAD\_COMMAND** EQU 1  
**TRACE\_CHILD\_NOT\_FOUND** EQU 2  
**TRACE\_CHILD\_UNTRACEABLE** EQU 5

If Ptrace\_B.Cmd is returned as TRC\_C\_SIG\_ret, the process is about to receive a signal.

If Ptrace\_B.Cmd is returned as TRC\_C\_KIL\_ret, the process is about to terminate.

If Ptrace\_B.Cmd returns as TRC\_C\_GPF\_ret, the process creates a General Protection fault. The fault type is returned in PTrace\_B.Value, and SegV:OffV contains the reference that generated the fault.

If Ptrace\_B.Cmd is returned as TRC\_C\_LIB\_ret, a library module has been loaded. The new module table entry (MTE) is returned in Ptrace\_B.Value. This can be used with the library support commands to identify the library module. The program module's MTE is returned in PTrace\_B.MTE. In this case, the initial TRC\_C\_Stop command should be re-issued until TRC\_C\_SUC\_ret is returned.

If Ptrace\_B.Cmd is returned as TRC\_C\_FPE\_ret, the process has generated a floating point error.

If Ptrace\_B.Cmd is returned as TRC\_C\_THD\_ret, the Ptrace\_b.TID field contains the thread ID of the terminating thread.

If Ptrace\_B.Cmd is returned as TRC\_C\_STP\_ret, then the asynchronous stop caused the debugger to stop.

# DosPtrace —

## Provides an Interface for Program Debugging

### C Language

```
#define INCL_DOSQUEUES

USHORT rc = DosPtrace(PtraceB);

PBYTE PtraceB; /* Ptrace buffer (returned) */

USHORT rc; /* return code */
```

### Assembler Language

```
EXTRN DosPtrace:FAR
INCL_DOSQUEUES EQU 1

PUSH@ OTHER Ptrace_B ;Ptrace buffer (returned)
CALL DosPtrace

Returns WORD
```

# DosPurgeQueue — Purge Queue

---

This call purges a queue of all elements.

<b>DosPurgeQueue</b> (QueueHandle)
------------------------------------

## Parameters

**QueueHandle** (*HQUEUE*) — input  
Handle of the queue to purge.

**rc** (*USHORT*) — return  
Return code descriptions are:

0	NO_ERROR
330	ERROR_QUE_PROC_NOT_OWNED
337	ERROR_QUE_INVALID_HANDLE

## Remarks

A process that creates a queue with `DosCreateQueue` owns it. Only the owning process and any threads it creates can issue `DosPurgeQueue` to remove all the elements from the queue.

## C Language

```
#define INCL_DOSQUEUES

USHORT rc = DosPurgeQueue(QueueHandle);

HQUEUE QueueHandle; /* Queue handle */

USHORT rc; /* return code */
```

## Assembler Language

```
EXTRN DosPurgeQueue:FAR
INCL_DOSQUEUES EQU 1

PUSH WORD QueueHandle ;Queue handle
CALL DosPurgeQueue

Returns WORD
```



## Output Message Text to Indicated Handle

This call outputs the message in a buffer passed by a caller to the specified handle. The function formats the buffer to prevent words from wrapping if displayed to a screen.

**DosPutMessage** (*FileHandle*, *MessageLength*, *MessageBuffer*)

### Parameters

**FileHandle** (*USHORT*) — input

Handle of the output file or device.

**MessageLength** (*USHORT*) — input

Length of the message to be output.

**MessageBuffer** (*PCHAR*) — input

Address of the buffer that contains the returned message.

**rc** (*USHORT*) — return

Return code descriptions are:

0	NO_ERROR
6	ERROR_INVALID_HANDLE
19	ERROR_WRITE_PROTECT
321	ERROR_MR_UN_PERFORM

### Remarks

Screen width is assumed to be 80 characters. The DosPutMessage call counts a CR/LF in the 80 characters that it tries to write to the screen. If a word extends past column 78, it is put on the next line.

DosPutMessage assumes the starting cursor position is column one when handling a word wrap.

If the last character to be positioned on a line is a double-byte character that would be bisected, the rule above ensures that the character is not bisected.

### C Language

```
#define INCL_DOSMISC
```

```
USHORT rc = DosPutMessage(FileHandle, MessageLength, MessageBuffer);
```

```
USHORT      FileHandle;    /* Handle of output file/device */
USHORT      MessageLength; /* Length of message buffer */
PCHAR       MessageBuffer; /* Message buffer */
```

```
USHORT      rc;            /* return code */
```

### Assembler Language

```
EXTRN DosPutMessage:FAR
INCL_DOSMISC EQU 1
```

```
PUSH WORD FileHandle ;Handle of output file/device
PUSH WORD MessageLength ;Length of message buffer
PUSH@ OTHER MessageBuffer ;Message buffer
CALL DosPutMessage
```

```
Returns WORD
```

# DosQAppType – Return the Application Type

This call returns the application type of an executable file.

**DosQAppType (ExecutableFileName, AppType)**

## Parameters

**ExecutableFileName (PSZ)** – input

Address of the ASCIIZ string containing the filename of the executable file where the flags are returned.

If the string appears to be a fully qualified path (contains a ":" in the second position and/or contains a "\"), only the indicated drive:directory is searched. If the string is not a fully qualified path, the current directory is searched. If the filename is not found in the current directory, each drive:directory specification in the PATH defined in the current program's environment is searched for this file. Note that any extension (.xxx) is acceptable for the executable filename. If no extension is specified, a default extension of ".exe" is used. AppType is a word that contains flags denoting the application type, as determined by reading the executable file header specified by ExecutableFileName. Note that the call sequence passes a pointer to a location in application memory to return the application type flags.

**AppType (PUSHORT)** – output

Address of the application type defined as follows:

Bit	Description										
15 – 6	Reserved.										
5	Set to 1 if executable file is PC DOS format. Bits 0, 1, 2, 3, and 4 are set to zero.										
4	Set to 1 if executable file is a dynamic link module. Bits 0, 1, 2, 3, and 5 are set to 0.										
3	Set to 1 if executable has been "bound" (BIND command) as a Family API application. Bits 0, 1 and 2 still apply.										
2 – 0	Bits 2, 1 and 0 indicate application type, as specified in the header of the executable file.										
	<table><tr><th>Value</th><th>Definition</th></tr><tr><td>000</td><td>Application type is not specified in executable header.</td></tr><tr><td>001</td><td>Application is NOTWINDOWCOMPAT</td></tr><tr><td>010</td><td>Application type is WINDOWCOMPAT</td></tr><tr><td>011</td><td>Application type is WINDOWAPI</td></tr></table>	Value	Definition	000	Application type is not specified in executable header.	001	Application is NOTWINDOWCOMPAT	010	Application type is WINDOWCOMPAT	011	Application type is WINDOWAPI
Value	Definition										
000	Application type is not specified in executable header.										
001	Application is NOTWINDOWCOMPAT										
010	Application type is WINDOWCOMPAT										
011	Application type is WINDOWAPI										

**rc (USHORT)** – return

Return code descriptions are:

0	NO_ERROR
2	ERROR_FILE_NOT_FOUND
3	ERROR_PATH_NOT_FOUND
4	ERROR_TOO_MANY_OPEN_FILES
11	ERROR_BAD_FORMAT
15	ERROR_INVALID_DRIVE
32	ERROR_SHARING_VIOLATION
108	ERROR_DRIVE_LOCKED
110	ERROR_OPEN_FAILED
191	ERROR_INVALID_EXE_SIGNATURE
192	ERROR_EXE_MARKED_INVALID

## Remarks

This function is used by the Presentation Manager shell to determine the application type being executed. The application type is specified at link time in the module definition file.

# DosQAppType – Return the Application Type

## C Language

```
#define INCL_DOSSESMGR

USHORT rc = DosQAppType(ExecutableFileName, AppType);

PSZ      ExecutableFileName; /* Address of executable file path name string */
PUSHORT  AppType;            /* Address to put application type flags */

USHORT    rc;                /* return code */
```

## Assembler Language

```
EXTRN DosQAppType:FAR
INCL_DOSSESMGR EQU 1

PUSH@ ASCIIZ ExecutableFileName ;Executable file path name string
PUSH@ WORD AppType;             ;Application type flags (returned)
CALL DosQAppType

Returns WORD
```

---

This call returns the full path name of the current directory for the requesting process for the specified drive.

<b>DosQCurDir</b> ( <i>DriveNumber</i> , <i>DirPath</i> , <i>DirPathLen</i> )
---

## Parameters

**DriveNumber** (*USHORT*) — input  
Drive number, for example:

Value	Definition
0	default
1	A
2	B
⋮	

**DirPath** (*PBYTE*) — output  
Address of the fully qualified path name of current directory.

**DirPathLen** (*PUSHORT*) — input/output  
Address of the length of the DirPath buffer. On input, this field contains the length of the directory path buffer in bytes. On output, if an error is returned because the buffer is too small, this field contains the required length.

**rc** (*USHORT*) — return  
Return code descriptions are:

0	NO_ERROR
15	ERROR_INVALID_DRIVE
26	ERROR_NOT_DOS_DISK
108	ERROR_DRIVE_LOCKED
111	ERROR_BUFFER_OVERFLOW

## Remarks

The drive letter is not part of the returned string. The string does not begin with a backslash and is terminated by a byte containing 00H.

The system returns the length of the returned DirPath string in DirPathLen, which does not include the terminating null byte. In the case where the DirPath buffer is of insufficient length to hold the current directory path string, the system returns the required length (in bytes) for DirPath in DirPathLen.

For FSDs, the case of the current directory is set according to the DirName passed in, not according to the case of the directories on disk. For example, if the directory "c:\bin" is created and DosChDir is called with DirName "c:\bin", the current directory returned by DosQCurDir will be "c:\bin".

Programs running without the NEWFILES bit set are allowed to DosChDir to a non-8.3 filename format directory.

DosQSysInfo must be used by an application to determine the maximum path length supported by OS/2. The returned value should be used to dynamically allocate buffers that are to be used to store paths.

# DosQCurDir — Query Current Directory

FAPI

## C Language

```
#define INCL_DOSFILEMGR

USHORT rc = DosQCurDir(DriveNumber, DirPath, DirPathLen);

USHORT DriveNumber; /* Drive number */
PBYTE DirPath; /* Directory path buffer (returned) */
PUSHORT DirPathLen; /* Directory path buffer length (returned) */

USHORT rc; /* return code */
```

## Assembler Language

```
EXTRN DosQCurDir:FAR
INCL_DOSFILEMGR EQU 1

PUSH WORD DriveNumber ;Drive number
PUSH@ OTHER DirPath ;Directory path buffer (returned)
PUSH@ WORD DirPathLen ;Directory path buffer length (returned)
CALL DosQCurDir

Returns WORD
```

This call determines the current default drive for the requesting process.

**DosQCurDisk (DriveNumber, LogicalDriveMap)**

## Parameters

**DriveNumber** (*PUSHORT*) – output

Address of the number of the default drive, for example:

Value	Definition
1	A
2	B
⋮	

**LogicalDriveMap** (*PULONG*) – output

Address of the bit map (stored in the low-order portion of the 32-bit, doubleword area) where the system returns the mapping of the logical drives. Logical Drives A to Z have a one-to-one mapping with the bit positions 0 to 25 of the map; for example, bit 0 is drive A, bit 1 is drive B, and so forth. The settings of these bits indicate which drives exist:

Value	Definition
0	The logical drive does not exist.
1	The logical drive exists.

**rc** (*USHORT*) – return

Return code description is:

0	NO_ERROR
---	----------

## C Language

```
#define INCL_DOSFILEMGR
```

```
USHORT rc = DosQCurDisk(DriveNumber, LogicalDriveMap);
```

```
PUSHORT DriveNumber; /* Default drive number (returned) */
PULONG LogicalDriveMap; /* Drive map area (returned) */
```

```
USHORT rc; /* return code */
```

## Assembler Language

```
EXTRN DosQCurDisk:FAR
```

```
INCL_DOSFILEMGR EQU 1
```

```
PUSH@ WORD DriveNumber ;Default drive number (returned)
PUSH@ DWORD LogicalDriveMap ;Drive map area (returned)
CALL DosQCurDisk
```

Returns WORD

---

This call queries the state of the specified file.

<b>DosQFHandState</b> ( <i>FileHandle</i> , <i>FileHandleState</i> )
--

## Parameters

**FileHandle** (*HFILE*) — input

Handle of the file to be queried.

**FileHandleState** (*PUSHORT*) — output

Address of the contents of the open mode word defined in a previous DosOpen or DosOpen2 call.

Bit	Description
15	Direct Open flag:  0 = PathName represents a file to be opened in the normal way.  1 = PathName is "Drive:" and represents a mounted disk or diskette volume to be opened for direct access.
14	Write-Through flag:  0 = Writes to the file may be run through the file system buffer cache.  1 = Writes to the file may go through the file system buffer cache but the sectors are written (actual file I/O completed) before a synchronous write call returns. This state of the file defines it as a synchronous file. For synchronous files, this is a mandatory bit in that the data must be written out to the medium for synchronous write operations.  This bit is not inherited by child processes.
13	Fail-Errors flag. Media I/O errors are handled as follows:  0 = Reported through the system critical error handler.  1 = Reported directly to the caller by return code.  Media I/O errors generated through an IOCTL Category 8 function always get reported directly to the caller by return code. The Fail-Errors function applies only to non-IOCTL handle-based file I/O calls.  This bit is not inherited by child processes.
12	No-Cache/Cache:  0 = It is advisable for the disk driver to cache the data in I/O operations on this file.  1 = I/O to the file need not be done through the disk driver cache.  This bit is an advisory bit, and is used to advise FSDs and device drivers on whether it is worth caching the data. This bit, like the write-through bit, is a per-handle bit.  This bit is not inherited by child processes.
11 – 8	Reserved Bits.
7	Inheritance flag:  0 = File handle is inherited by a spawned process resulting from a DosExecPgm call.  1 = File handle is private to the current process.  This bit is not inherited by child processes.

**6—4**      **Sharing Mode flags:** The file sharing mode flags define what operations other processes may perform on the file shown as follows:

Value	Definition
001	Deny Read/Write access
010	Deny Write access
011	Deny Read access
100	Deny neither Read or Write access (Deny None).

Any other value is invalid.

**3**      **Reserved Bit.**

**2—0**      **Access Mode flags.** The file access is assigned as follows:

Value	Definition
000	Read-Only access
001	Write-Only access
010	Read/Write access.

Any other value is invalid.

**rc (USHORT) — return**

Return code descriptions are:

0	NO_ERROR
6	ERROR_INVALID_HANDLE

## Remarks

When a critical error occurs that the application cannot handle, it may reset critical error handling to be performed by the system. This is done by calling `DosSetFHandState` to turn off the fail/errors bit and then reissuing the I/O call. The expected critical error reoccurs, and control is passed to the system critical error handler. The precise time that the effect of this function is visible at the application level is unpredictable when asynchronous I/O is pending.

The DASD Open bit parameter is the "Direct I/O flag." It provides an access mechanism to a disk or diskette volume independent of the file system. This mode should only be used by systems programs and not by application programs.

## Named Pipe Considerations

As defined by OS/2 D = 0. Other bits as defined by `DosMakeNmPipe` (serving end), `DosOpen` (client end), or the last `DosSetFHandState`.

## C Language

```
#define INCL_DOSFILEMGR
```

```
USHORT rc = DosQFHandState(FileHandle, FileHandleState);
```

```
HFILE      FileHandle;      /* File handle */
PUSHORT    FileHandleState; /* File handle state (returned) */
```

```
USHORT      rc;              /* return code */
```

## Assembler Language

```
EXTRN DosQFHandState:FAR
INCL_DOSFILEMGR EQU 1
```

```
PUSH WORD FileHandle ;File handle
PUSH@ WORD FileHandleState ;File handle state (returned)
CALL DosQFHandState
```

Returns WORD



# DosQFileInfo — Query File Information

FAP1

This call returns information for a specific file.

**DosQFileInfo** (**FileHandle**, **FileInfoLevel**, **FileInfoBuf**, **FileInfoBufSize**)

## Parameters

**FileHandle** (*HFILE*) — input  
File handle.

**FileInfoLevel** (*USHORT*) — input  
Level of file information required. A value of 1, 2, or 3 can be specified. Level 4 is reserved. The structures described in **FileInfoBuf** indicate the information returned for each of these levels.

**FileInfoBuf** (*PBYTE*) — output  
Address of the storage area where the system returns the requested level of file information. File information, where applicable, is at least as accurate as the most recent **DosClose**, **DosBufReset**, **DosSetFileInfo**, or **DosSetPathInfo**.

### Level 1 Information

**FileInfoBuf** contains the following structure, to which file information is returned:

#### **filedate** (*FDATE*)

Structure containing the date of file creation.

Bit	Description
15–9	Year, in binary, of file creation
8–5	Month, in binary, of file creation
4–0	Day, in binary, of file creation.

#### **filetime** (*FTIME*)

Structure containing the time of file creation.

Bit	Description
15–11	Hours, in binary, of file creation
10–5	Minutes, in binary, of file creation
4–0	Seconds, in binary number of two-second increments, of file creation.

#### **fileaccessdate** (*FDATE*)

Structure containing the date of last access. See *FDATE* in **filedate**.

#### **fileaccessstime** (*FTIME*)

Structure containing the time of last access. See *FTIME* in **filetime**.

#### **writeaccessdate** (*FDATE*)

Structure containing the date of last write. See *FDATE* in **filedate**.

#### **writeaccessstime** (*FTIME*)

Structure containing the time of last write. See *FTIME* in **filetime**.

#### **filesize** (*ULONG*)

File size.

#### **filealloc** (*ULONG*)

Allocated file size.

#### **fileattrib** (*USHORT*)

Attributes of the file, defined in **DosSetFileMode**.

### Level 2 Information

**FileInfoBuf** contains a structure similar to the Level 1 structure, with the addition of the **cbList** field after the **fileattrib** field.

**cbList (ULONG)**

On output, this field contains the length of the entire EA set for the file object. This value can be used to calculate the size of the buffer required to hold EA information returned when FileInfoLevel = 3 is specified.

**Level 3 Information**

FileInfoBuf contains an EAOP structure, which has the following format:

**fpGEAList (PGEALIST)**

Address of GEAList. GEAList is a packed array of variable length "get EA" structures, each containing an EA name and the length of the name.

**fpFEAList (PFEALIST)**

Address of FEAList. FEAList is a packed array of variable length "full EA" structures, each containing an EA name and its corresponding value, as well as the lengths of the name and the value.

**oError (ULONG)**

Offset into structure where error has occurred.

On input, FileInfoBuf is an EAOP structure. fpGEAList points to a GEA list defining the attribute names whose values are returned. fpFEAList points to a data area where the relevant FEA list is returned. The length field of this FEA list is valid, giving the size of the FEA list buffer. oError points to the offending GEA entry in case of error. Following is the format of the GEAList structure:

**cbList (ULONG)**

Length of the GEA list, including the length itself.

**list (GEA)**

List of GEA structures. A GEA structure has the following format:

**cbName (BYTE)**

Length of EA ASCIIZ name, which does not include the null character.

**szName (CHAR)**

ASCIIZ name of EA.

On output, FileInfoBuf is unchanged. The buffer pointed to by fpFEAList is filled in with the returned information. If the buffer fpFEAList points to isn't large enough to hold the returned information (ERROR\_BUFFER\_OVERFLOW) cbList is still valid, assuming there's at least enough space for it. Its value is the size of the entire EA set for the file, even though only a subset of attributes was requested. Following is the format of the FEAList structure:

**cbList (ULONG)**

Length of the FEA list, including the length itself.

**list (FEA)**

List of FEA structures. An FEA structure has the following format:

**Flags (BYTE)**

Bit indicator describing the characteristics of the EA being defined.

Bit	Description
15	Critical EA.
14 – 0	Reserved and must be set to zero.

If bit 15 is set to 1, this indicates a critical EA. If bit 15 is 0, this means the EA is noncritical; that is, the EA is not essential to the intended use by an application of the file with which it is associated.

**cbName (BYTE)**

Length of EA ASCIIZ name, which does not include the null character.

**cbValue (USHORT)**

Length of EA value, which cannot exceed 64KB.

# DosQFileInfo — Query File Information

FAPI

**szName** (*PSZ*)

Address of the ASCIIZ name of EA.

**aValue** (*PSZ*)

Address of the free-format value of EA.

**Note:** The szName and aValue fields are not included as part of header or include files. Because of their variable lengths, these entries must be built manually.

**FileInfoBufSize** (*USHORT*) — output  
Length of FileInfoBuf.

**rc** (*USHORT*) — return  
Return code descriptions are:

0	NO_ERROR
5	ERROR_ACCESS_DENIED
6	ERROR_INVALID_HANDLE
111	ERROR_BUFFER_OVERFLOW
124	ERROR_INVALID_LEVEL
130	ERROR_DIRECT_ACCESS_HANDLE
254	ERROR_INVALID_EA_NAME
255	ERROR_EA_LIST_INCONSISTENT

## Remarks

The FAT file system supports modification of only date and time information contained in file information level 1. Zero is returned for the creation and access dates and times.

To return information contained in any of the file information levels, DosQFileInfo has to be able to read the open file. DosQFileInfo works only when the file is opened for read access, with a deny-write sharing mode specified for access by other processes. If the file is already opened by another process that has specified conflicting sharing and access modes, a call to DosOpen or DosOpen2 fails.

DosEnumAttribute returns the lengths of EAs. This information can be used to calculate the size of FileInfoBuf needed to hold full extended attribute (FEA) information returned by DosQFileInfo when Level 3 is specified. The size of the buffer is calculated as follows:

One byte (for fea\_usFlags) +  
One byte (for fea\_cbName) +  
Two bytes (for fea\_cbValue) +  
Value of cbName (for the name of the EA) +  
One byte (for terminating NULL in fea\_cbName) +  
Value of cbValue (for the value of the EA)

## C Language

```
typedef struct _FDATE {    /* fdate */

    unsigned day   : 5;    /* binary day for directory entry */
    unsigned month : 4;    /* binary month for directory entry */
    unsigned year  : 7;    /* binary year for directory entry */

} FDATE;

typedef struct _FTIME {    /* ftime */

    unsigned twosecs : 5;    /* binary number of two-second increments */
    unsigned minutes : 6;    /* binary number of minutes */
    unsigned hours   : 5;    /* binary number of hours */

} FTIME;
```

```

typedef struct _FILESTATUS {    /* fsts */

    FDATE  fdateCreation;      /* date of file creation */
    FTIME  ftimeCreation;      /* time of file creation */
    FDATE  fdateLastAccess;    /* date of last access */
    FTIME  ftimeLastAccess;    /* time of last access */
    FDATE  fdateLastWrite;     /* date of last write */
    FTIME  ftimeLastWrite;     /* time of last write */
    ULONG  cbFile;             /* file size (end of data) */
    ULONG  cbFileAlloc;        /* file allocated size */
    USHORT attrFile;           /* attributes of the file */

} FILESTATUS;

typedef struct _GEA {          /* gea */

    BYTE  cbName;              /* name length not including NULL */
    CHAR  szName[1];           /* attribute name */

} GEA;

typedef struct _GEALIST {     /* geal */

    ULONG  cbList;             /* total bytes of structure including full list */
    GEA  list[1];              /* variable length GEA structures */

} GEALIST;

typedef struct _FEA {         /* fea */

    BYTE  fEA;                 /* flags */
    BYTE  cbName;              /* name length not including NULL */
    USHORT cbValue;            /* value length */

} FEA;

typedef struct _FEALIST {     /* feal */

    ULONG  cbList;             /* total bytes of structure including full list */
    FEA  list[1];              /* variable length FEA structures */

} FEALIST;

typedef struct _EAOP {        /* eaop */

    PGEALIST fpGEAList;        /* general EA list */
    PFEALIST fpFEAList;        /* full EA list */
    ULONG  oError;

} EAOP;

#define INCL_DOSFILEMGR

USHORT  rc = DosQFileInfo(FileHandle, FileInfoLevel, FileInfoBuf,
                          FileInfoBufSize);

HFILE      FileHandle;        /* File handle */
USHORT      FileInfoLevel;     /* File data required */
PBYTE      FileInfoBuf;       /* File data buffer (returned) */
USHORT      FileInfoBufSize;   /* File data buffer size */

USHORT      rc;                /* return code */

```

## Assembler Language

# DosQFileInfo — Query File Information

FAPI

```
FDATE    struc
    fdate_fs dw ?
FDATE    ends

FTIME    struc
    ftime_fs dw ?
FTIME    ends

FILESTATUS struc
    fsts_fdateCreation dw (size FDATE)/2 dup (?) ;date of file creation
    fsts_ftimeCreation dw (size FTIME)/2 dup (?) ;time of file creation
    fsts_fdateLastAccess dw (size FDATE)/2 dup (?) ;date of last access
    fsts_ftimeLastAccess dw (size FTIME)/2 dup (?) ;time of last access
    fsts_fdateLastWrite dw (size FDATE)/2 dup (?) ;date of last write
    fsts_ftimeLastWrite dw (size FTIME)/2 dup (?) ;time of last write
    fsts_cbFile dd ? ;file size (end of data)
    fsts_cbFileAlloc dd ? ;file allocated size
    fsts_attrFile dw ? ;attributes of the file
FILESTATUS ends

GEA    struc
    gea_cbName db ? ;name length not including NULL
    gea_szName db 1 dup (?) ;attribute name
GEA    ends

GEALIST    struc
    geal_cbList dd ? ;total bytes of structure including full list
    geal_list db size GEA * 1 dup (?) ;variable length GEA structures
GEALIST    ends

FEA    struc
    fea_fEA db ? ;flags
    fea_cbName db ? ;name length not including NULL
    fea_cbValue dw ? ;value length
FEA    ends

FEALIST    struc
    feal_cbList dd ? ;total bytes of structure including full list
    feal_list db size FEA * 1 dup (?) ;variable length FEA structures
FEALIST    ends

EAOP    struc
    eaop_fpGEAList dd ? ;general EA list
    eaop_fpFEAList dd ? ;full EA list
    eaop_oError dd ? ;
EAOP    ends

EXTRN DosQFileInfo:FAR
INCL_DOSFILEMGR EQU 1

PUSH WORD FileHandle ;File handle
PUSH WORD FileInfoLevel ;File data required
PUSH@ OTHER FileInfoBuf ;File data buffer (returned)
PUSH WORD FileInfoBufSize ;File data buffer size
CALL DosQFileInfo

Returns WORD
```

---

This call queries the mode (attribute) of the specified file.

<b>DosQFileMode</b> ( <b>FilePathName</b> , <b>CurrentAttribute</b> , <b>Reserved</b> )
---

## Parameters

**FilePathName** (*PSZ*) – input

Address of the file path name.

DosQSysInfo is called by an application during initialization to determine the maximum path length allowed by OS/2.

**CurrentAttribute** (*PUSHORT*) – output

Address of the file's current attribute.

Bit	Description
15 – 6	Reserved.
5	File archive
4	Subdirectory
3	Reserved.
2	System file
1	Hidden file
0	Read only file

These bits can be set individually or in combination. For example, an attribute value of 0021H (bits 5 and 0 set to 1) indicates a read-only file that is archived.

**Reserved** (*ULONG*) – input

Reserved must be set to zero.

**rc** (*USHORT*) – return

Return code descriptions are:

0	NO_ERROR
2	ERROR_FILE_NOT_FOUND
3	ERROR_PATH_NOT_FOUND
26	ERROR_NOT_DOS_DISK
87	ERROR_INVALID_PARAMETER
108	ERROR_DRIVE_LOCKED
206	ERROR_FILENAME_EXCED_RANGE

## Remarks

The 'Volume Label' type attribute is not returned by DosQFileMode. DosQFSInfo may be used for this purpose.

## C Language

```
#define INCL_DOSFILEMGR
```

```
USHORT rc = DosQFileMode(FilePathName, CurrentAttribute, Reserved);
```

```
PSZ      FilePathName;    /* File path name */
PUSHORT  CurrentAttribute; /* Data area (returned) */
ULONG    0;               /* Reserved (must be zero) */

USHORT    rc;             /* return code */
```

# DosQFileMode — Query File Mode

FAP1

## Assembler Language

```
EXTRN DosQFileMode:FAR
INCL_DOSFILEMGR      EQU 1

PUSH@ ASCIIZ  FilePathName      ;File path name
PUSH@ WORD    CurrentAttribute   ;Data area (returned)
PUSH  DWORD   0                  ;Reserved (must be zero)
CALL  DosQFileMode
```

Returns WORD

# DosQFSAttach – Query Attached FSD Information

Query information about an attached file system (local or remote), or about a character device or pseudo-character device attached to the file system.

<b>DosQFSAttach</b> ( <b>DeviceName</b> , <b>Ordinal</b> , <b>FSAInfoLevel</b> , <b>DataBuffer</b> , <b>DataBufferLen</b> , <b>Reserved</b> )
---

## Parameters

**DeviceName** (*PSZ*) – input

Address of a drive letter or the name of a character or pseudo-character device. If **DeviceName** is a drive, it is an ASCIIZ string having the form of drive letter followed by a colon. If **DeviceName** is a character or pseudo-character device name, its format is that of an ASCIIZ string in the format of an OS/2 file name in a subdirectory called \DEV\. This parameter is ignored if level 2 or 3 is specified for **FSAInfoLevel**.

**Ordinal** (*USHORT*) – output

Index into the list of character or pseudo-character devices, or the set of drives. **Ordinal** always starts at 1. The **Ordinal** position of an item in a list has no significance at all. **Ordinal** is used strictly to step through the list. The mapping from **Ordinal** to item is volatile and may change from one call to **DosQFSAttach** to the next. This parameter is ignored if level 1 is specified for **FSAInfoLevel**.

**FSAInfoLevel** (*USHORT*) – input

Level of information returned in **DataBuffer**:

- Level 0001H returns data for the specific drive or device name referred to by **DeviceName**. The **Ordinal** field is ignored.
- Level 0002H returns data for the entry in the list of character or pseudo-character devices selected by **Ordinal**. The **DeviceName** field is ignored.
- Level 0003H returns data for the entry in the list of drives selected by **Ordinal**. The **DeviceName** field is ignored.

**DataBuffer** (*PBYTE*) – output

Address of the return information buffer, has the following format:

**IType** (*USHORT*)

Type of item.

Value	Definition
1	Resident character device
2	Pseudo-character device
3	Local drive
4	Remote drive attached to FSD.

**cbName** (*USHORT*)

Length of item name, not counting null.

**szName** (*UCHAR*)

Item name, ASCIIZ string.

**cbFSDName** (*USHORT*)

Length of FSD name, not counting null.

**szFSDName** (*UCHAR*)

Name of FSD item is attached to, ASCIIZ string.

**cbFSADData** (*USHORT*)

Length of FSD Attach data returned by FSD.



# DosQFSAttach –

## Query Attached FSD Information

### **rgFSAData** (*UCHAR*)

FSD Attach data returned by FSD.

**Note:** The szFSDName is the FSD name exported by the FSD, which is not necessarily the same as the FSD name in the boot sector.

For local character devices (*iType* = 1), *cbFSDName* = 0 and *szFSDName* contains only a terminating NULL byte, and *cbFSAData* = 0.

For local drives (*iType* = 3), *szFSDName* contains the name of the FSD attached to the drive at the time of the call. This information changes dynamically. If the drive is attached to the kernel's resident file system, *szFSDName* contains FAT or unknown. Since the resident file system gets attached to any disk that other FSDs refuse to mount, it is possible to have a disk that does not contain a recognizable file system, but yet gets attached to the resident file system. In this case, it is possible to detect the difference, and this information would help programs in not destroying data on a disk that was not properly recognized.

### **DataBufferLen** (*PUSHORT*) – output

Address of the byte length of the return buffer. Upon return, it is the length of the data returned in *DataBuffer* by the FSD.

### **Reserved** (*ULONG*) – input

Reserved, must be set to zero.

### **rc** (*USHORT*) – return

Return code descriptions are:

0	NO_ERROR
15	ERROR_INVALID_DRIVE
111	ERROR_BUFFER_OVERFLOW
124	ERROR_INVALID_LEVEL
259	ERROR_NO_MORE_ITEMS

## Remarks

Information about all block devices and all character and pseudo-character devices is returned by this call.

The information returned by this call is highly volatile. Calling programs should be aware that the returned information may have already changed by the time it's returned to them.

The information returned for disks that are attached to the kernel's resident file system can be used to determine if the kernel definitely recognized the disk as one with its file system on it, or if the kernel just attached its file system to it because no other FSDs mounted the disk. This can be important information for a program that needs to know what file system is attached to the drive. It is quite easy to get into a situation where the FSD that recognizes a certain disk has not been installed into the system. In such a case, there is a potential for the data on the disk to be destroyed since the wrong file system is attached to the disk by default.

## C Language

```
#define INCL_DOSFILEMGR
```

```
USHORT rc = DosQFSAttach(DeviceName, Ordinal, FSAInfoLevel, DataBuffer, DataBufferLen, 0);
```

```
PSZ      DeviceName;    /* Device name or drive letter string */
USHORT   Ordinal;       /* Ordinal of entry in name list */
USHORT   FSAInfoLevel;  /* Type of attached FSD data required */
PBYTE    DataBuffer;    /* Returned data buffer */
PUSHORT  DataBufferLen; /* Buffer length */
ULONG    0;             /* Reserved (must be zero) */

USHORT    rc;           /* return code */
```

## DosQFSAttach — Query Attached FSD Information

### Assembler Language

```
EXTRN DosQFSAttach:FAR
INCL_DOSFILEMGR      EQU 1

PUSH@  ASCIIZ DeviceName      ;Device name or drive letter string
PUSH   WORD   Ordinal         ;Ordinal of entry in name list
PUSH   WORD   FSAInfoLevel    ;Type of attached FSD data required
PUSH@  OTHER  DataBuffer      ;Data buffer (returned)
PUSH@  WORD   DataBufferLen    ;Buffer length (returned)
PUSH   DWORD  0               ;Reserved (must be zero)
CALL   DosQFSAttach
```

Returns WORD

# DosQFSInfo – Query File System Information

FAPI

This call queries information from a file system device.

**DosQFSInfo (DriveNumber, FSInfoLevel, FSInfoBuf, FSInfoBufSize)**

## Parameters

**DriveNumber** (*USHORT*) – input

Logical drive number (0 = default, 1 = A, and so on).

When a logical drive is specified, the media in the drive is examined (local drive only) and the request is passed to the FSD responsible for managing that media or to the FSD that is attached to the drive.

**FSInfoLevel** (*USHORT*) – input

Level of file information required.

**FSInfoBuf** (*PBYTE*) – output

Address of the storage area where the system returns the requested level of file information.

### Level 1 Information

For FSInfoLevel = 1, information is returned in the following structure:

**filesysid** (*ULONG*)

File system ID.

**sectornum** (*ULONG*)

Number of sectors per allocation unit.

**unitnum** (*ULONG*)

Number of allocation units.

**unitavail** (*ULONG*)

Number of allocation units available.

**bytesnum** (*USHORT*)

Number of bytes per sector.

### Level 2 Information

For FSInfoLevel = 2, the information is returned in the following format:

**reserved** (*ULONG*)

Reserved

**volumelength** (*BYTE*)

Length of the volume label, not including the null.

**volumelabel** (*CHAR*)

Volume label ASCII string.

**FSInfoBufSize** (*USHORT*)

Length of the buffer.

**rc** (*USHORT*) – return

Return code descriptions are:

0	NO_ERROR
15	ERROR_INVALID_DRIVE
111	ERROR_BUFFER_OVERFLOW
124	ERROR_INVALID_LEVEL
125	ERROR_NO_VOLUME_LABEL

## Remarks

Trailing blanks supplied at volume label definition time are not considered to be part of the label and are therefore not returned as valid label data. Volume label is limited to a length of 11 bytes.

Volume Serial Number is a unique 32-bit number used by OS/2 to positively identify its disk/diskette volumes. The hard error prompts the user for an unmounted removable volume by displaying both the Volume Serial Number (as an 8 digit hexadecimal number) and the Volume Label.

If there is no volume serial number on the disk/diskette, the volume serial number information is returned as binary zeros. If there is no volume label on the disk/diskette, the volume label information is returned as blank spaces. If there is no volume serial number and/or volume label for disk/diskette volumes formatted by DOS 3.X, this information is not displayed by the Hard Error Handler.

## C Language

```
typedef struct _FSALLOCATE {
    ULONG idFileSystem; /* file system ID */
    ULONG cSectorUnit; /* number sectors per allocation unit */
    ULONG cUnit; /* number of allocation units */
    ULONG cUnitAvail; /* available allocation units */
    USHORT cbSector; /* bytes per sector */
} FSALLOCATE;

typedef struct _FDATE { /* fdate */

    unsigned day : 5; /* binary day for directory entry */
    unsigned month : 4; /* binary month for directory entry */
    unsigned year : 7; /* binary year for directory entry */

} FDATE;

typedef struct _FTIME { /* ftime */

    unsigned twosecs : 5; /* binary number of two-second increments */
    unsigned minutes : 6; /* binary number of minutes */
    unsigned hours : 5; /* binary number of hours */

} FTIME;

typedef struct _FSINFO { /* fsinf */
    FDATE fdateCreation;
    FTIME ftimeCreation;
    VOLUMELABEL vol;
} FSINFO;

typedef struct _VOLUMELABEL { /* vol */
    BYTE cch;
    CHAR szVolLabel[12];
} VOLUMELABEL;

#define INCL_DOSFILEMGR

USHORT rc = DosQFSInfo(DriveNumber, FSInfoLevel, FSInfoBuf, FSInfoBufSize);

USHORT DriveNumber; /* Drive number */
USHORT FSInfoLevel; /* File system data required */
PBYTE FSInfoBuf; /* File system info buffer */
USHORT FSInfoBufSize; /* File system info buffer size */

USHORT rc; /* return code */
```

# DosQFSInfo — Query File System Information

FAPI

## Assembler Language

```
FSALLOCATE struc
    fsalloc_idFileSystem      dd ?
    fsalloc_cSectorUnit      dd ?
    fsalloc_cUnit            dd ?
    fsalloc_cUnitAvail       dd ?
    fsalloc_cbSector         dw ?
FSALLOCATE ends

EXTRN DosQFSInfo:FAR
INCL_DOSFILEMGR      EQU 1

PUSH WORD DriveNumber ;Drive number
PUSH WORD FSInfoLevel ;File system data required
PUSH@ OTHER FSInfoBuf ;File system info buffer (returned)
PUSH WORD FSInfoBufSize ;File system info buffer size
CALL DosQFSInfo

Returns WORD
```

This call determines whether a handle references a file or a device.

**DosQHandType** (FileHandle, HandType, FlagWord)

## Parameters

**FileHandle** (*HFILE*) – input

File handle.

**HandType** (*PUSHORT*) – output

Address of the value indicating the handle type. HandType is composed of two bytes:

Bit	Description								
15	Network bit:  0 = The handle refers to a local file, device, or pipe. 1 = Means that the handle refers to a remote file, device, or pipe.								
14 – 8	Reserved.								
7 – 0	HandleClass describes the handle class. It may take on the following values in the low byte of HandType: <table data-bbox="374 945 869 1066"> <tr> <th>Value</th><th>Definition</th></tr> <tr> <td>0</td><td>Handle is for a disk file</td></tr> <tr> <td>1</td><td>Handle is for a character device</td></tr> <tr> <td>2</td><td>Handle is for a pipe.</td></tr> </table> Values greater than 2 are reserved.	Value	Definition	0	Handle is for a disk file	1	Handle is for a character device	2	Handle is for a pipe.
Value	Definition								
0	Handle is for a disk file								
1	Handle is for a character device								
2	Handle is for a pipe.								

**FlagWord** (*PUSHORT*) – output

Address of the device driver's attribute word if HandType indicates a local character device.

**rc** (*USHORT*) – return

Return code descriptions are:

0	NO_ERROR
6	ERROR_INVALID_HANDLE

## Remarks

This function allows programs that are interactive or file-oriented to determine the source of their input. For example, CMD.EXE suppresses writing prompts if the input is from a disk file.

## C Language

```
#define INCL_DOSFILEMGR
```

```
USHORT rc = DosQHandType(FileHandle, HandType, FlagWord);
```

```
HFILE      FileHandle;    /* File handle */
PUSHORT    HandType;      /* Handle type (returned) */
PUSHORT    FlagWord;      /* Device driver attribute (returned) */

USHORT     rc;            /* return code */
```

# DosQHandType — Query Handle Type

FAP1

## Assembler Language

```
EXTRN DosQHandType:FAR
INCL_DOSFILEMGR      EQU 1

PUSH  WORD    FileHandle    ;File handle
PUSH@ WORD    HandType      ;Handle type (returned)
PUSH@ WORD    FlagWord      ;Device driver attribute (returned)
CALL  DosQHandType
```

Returns WORD

## DosQNmPHandState – Query Named Pipe State

This call returns information for a named pipe's specific handle state.

<b>DosQNmPHandState</b> (Handle, PipeHandleState)
---

### Parameters

**Handle** (*HPIPE*) – input

Handle of the named pipe returned by DosMakeNmPipe or DosOpen.

**PipeHandleState** (*PUSHORT*) – output

Address of the named pipe handle state:

Bit	Description								
15	Blocking; pipe is defined as follows:  0 = Reads/Writes block if no data available.  1 = Reads/Writes return immediately if no data available.  Reads normally block until at least partial data can be returned. Writes, by default, block until all bytes requested have been written. Non-blocking mode (value is 1) changes this behavior as follows: <ul style="list-style-type: none"><li>• DosRead returns immediately with BytesRead = 0 (as defined in DosRead) if no data is available.</li><li>• DosWrite returns immediately with BytesWritten = 0 (as defined in DosWrite) if insufficient buffer space is available in the pipe, or the entire data area is transferred.</li></ul>								
14	Server or requester end; end-point of named pipe shown as follows:  0 = Handle is the client end of a named pipe.  1 = Handle is the server (requester) end of a named pipe.								
13 – 12	Reserved.								
11 – 10	Type of named pipe:  00 = Pipe is a byte stream pipe.  01 = Pipe is a message stream pipe.								
9 – 8	Read mode:  00 = Read pipe as a byte stream.  01 = Read pipe as a message stream.								
7 – 0	Instance count; 8-bit count to control pipe instances. When making the first instance of a named pipe, this field specifies how many instances can be created. Instances are as follows: <table><tr><th>Value</th><th>Definition</th></tr><tr><td>1</td><td>This can be the only instance (pipe is unique).</td></tr><tr><td>–1</td><td>The number of instances is unlimited.</td></tr><tr><td>0</td><td>Reserved value.</td></tr></table>	Value	Definition	1	This can be the only instance (pipe is unique).	–1	The number of instances is unlimited.	0	Reserved value.
Value	Definition								
1	This can be the only instance (pipe is unique).								
–1	The number of instances is unlimited.								
0	Reserved value.								

**rc** (*USHORT*) – return

Return code descriptions are:

0	NO_ERROR
230	ERROR_BAD_PIPE
233	ERROR_PIPE_NOT_CONNECTED



# DosQNmPHandState — Query Named Pipe State

## Remarks

At the serving end, the values returned by DosQNmPHandState are those originally established by DosMakeNmPipe or a subsequent DosSetNmPHandState. The values returned by the client end are those originally established by DosOpen or a subsequent DosSetNmPHandState.

## C Language

```
#define INCL_DOSNMPIPES

USHORT rc = DosQNmPHandState(Handle, PipeHandleState);

HPIPE      Handle;          /* Pipe handle */
PUSHORT     PipeHandleState; /* Pipe handle state */

USHORT      rc;              /* return code */
```

## Assembler Language

```
EXTRN DosQNmPHandState:FAR
INCL_DOSNMPIPES EQU 1

PUSH WORD Handle ;Pipe handle
PUSH@ WORD PipeHandleState ;Pipe handle state (returned)
CALL DosQNmPHandState

Returns WORD
```

# DosQNmPipeInfo – Query Named Pipe Info

---

This call returns information for a named pipe.

<b>DosQNmPipeInfo</b> ( <i>Handle</i> , <i>InfoLevel</i> , <i>InfoBuf</i> , <i>InfoBufSize</i> )
--

## Parameters

**Handle** (*HPIPE*) – input

Handle of the named pipe that is returned by `DosMakeNmPipe` or `DosOpen`.

**InfoLevel** (*USHORT*) – input

Level of the required pipe information. Only level 1 file information is supported.

**InfoBuf** (*PBYTE*) – output

Address of the storage area that returns the requested level of named pipe information. For `InfoLevel = 1`, file information is returned in the following format:

**outbufsize** (*USHORT*)

Actual size of buffer for outgoing I/O.

**inbufsize** (*USHORT*)

Actual size of buffer for ingoing I/O.

**maxnuminstances** (*UCHAR*)

Maximum allowed number of pipe instances.

**numinstances** (*UCHAR*)

Current number of pipe instances.

**namelength** (*UCHAR*)

Length of pipe name.

**pipename** (*CHAR*)

Name of pipe (including `\\ComputerName`, if remote).

**InfoBufSize** (*USHORT*) – input

Length of `InfoBuf`.

**rc** (*USHORT*) – return

Return code descriptions are:

0	NO_ERROR
111	ERROR_BUFFER_OVERFLOW
124	ERROR_INVALID_LEVEL
230	ERROR_BAD_PIPE

## Remarks

`DosQNmPipeInfo` returns all the information that fits in `InfoBuf`.

For level 1 information, if the length of the name of the pipe is greater than 255 bytes, then a length of zero is returned in the length field. The full ASCIIZ name will still be returned in this case.

# DosQNmPipeInfo — Query Named Pipe Info

## C Language

```
typedef struct  npidata {      /* PipeInfo data block (returned, level 1) */
    USHORT npioflen;          /* length of outgoing I/O buffer */
    USHORT npibflen;          /* length of incoming I/O buffer */
    UCHAR  npimaxcnt;         /* maximum number of instances */
    UCHAR  npicurcnt;         /* current number of instances */
    UCHAR  npinamlen;         /* length of pipe name */
    CHAR   npiname[1];        /* start of name */
}; /* npidata */
```

```
#define INCL_DOSNMPIPES
```

```
USHORT  rc = DosQNmPipeInfo(Handle, InfoLevel, InfoBuf, InfoBufSize);
```

```
HPIPE      Handle;          /* Pipe handle */
USHORT      InfoLevel;       /* Pipe data required */
PBYTE       InfoBuf;         /* Pipe data buffer */
USHORT      InfoBufSize;     /* Pipe data buffer size */

USHORT      rc;              /* return code */
```

## Assembler Language

```
EXTRN  DosQNmPipeInfo:FAR
INCL_DOSNMPIPES    EQU 1
```

```
PUSH  WORD  Handle      ;Pipe handle
PUSH  WORD  InfoLevel   ;Pipe data required
PUSH@ OTHER InfoBuf     ;Pipe data buffer
PUSH  WORD  InfoBufSize ;Pipe data buffer size
CALL  DosQNmPipeInfo
```

```
Returns WORD
```

# DosQNmPipeSemState – Query Named Pipe Operations

This call returns information about local named pipes attached to a specific system semaphore.

<b>DosQNmPipeSemState</b> (SemHandle, InfoBuf, InfoBufLen)
--

## Parameters

**SemHandle** (*HSEM*) – input

System semaphore handle that was previously attached to a named pipe by DosSetNmPipeSem.

**InfoBuf** (*PBYTE*) – output

Address of the buffer that contains records, or multiple records, for each named pipe:

**pipestatus** (*UCHAR*)

Coded value indicating the state of the named pipe:

Value	Definition
0	End of information buffer (EOF). No more information records follow and subsequent fields in this information record have no defined value.
1	Read data available.
2	Write space available.
3	Pipe is closed.

**pipestate** (*UCHAR*)

Bit mask indicating additional information about the state of the named pipe:

Bit	Description
7–1	Reserved
0	A thread is waiting on the other end of the pipe.

**keyhandle** (*USHORT*)

Key value associated with SemHandle at the time of the call to DosSetNmPipeSem.

**pipedata** (*USHORT*)

Pipe “data or space” availability state:

Value	Definition
1	Amount of read space available in the pipe.
2	Amount of write data available in the pipe.

**InfoBufLen** (*USHORT*) – input

Size in bytes of InfoBuf.

**rc** (*USHORT*) – return

Return code descriptions are:

0	NO_ERROR
87	ERROR_INVALID_PARAMETER
111	ERROR_BUFFER_OVERFLOW

## Remarks

A record is placed in InfoBuf for each local named pipe that has a semaphore attached whose handle matches the handle specified and whose state is closed or allows blocking mode I/O to be done.

There is no guarantee that records in the buffer refer to named pipes opened by the process making this call. If the same system semaphore is attached to different named pipes by multiple processes, information about named pipes that are not accessible to the caller can be returned. Thus, cooperating processes should agree on a convention for key values to help identify the named pipes of interest. A key value is associated with the pipe at the time the semaphore is set with DosSetNmPipeSem.

If a process wants data in the buffer to refer only to its own named pipes, it must use an exclusive system semaphore.

## DosQNmPipeSemState — Query Named Pipe Operations

A process associates a single semaphore with multiple pipes by way of `DosSetNmPipeSem`. After waking up from a wait on the semaphore, a thread issues `DosQNmPipeSemState`, which returns the I/O state information for all pipes associated with the semaphore. The thread can scan this information to determine which pipes can be read or written. This is more efficient than polling the pipes with a non-blocking I/O on each pipe.

### C Language

```
typedef struct npss {          /* QNmPipeSemState information record */
    UCHAR npss_status;        /* type of record, 0=EOL, 1=read ok, */
                              /* 2 = write ok, 3 = pipe closed */
    UCHAR npss_flag;          /* additional info, 01=waiting thread */
    USHORT npss_key;          /* user's key value */
    USHORT npss_avail;        /* available data/space if status=1/2 */
}; /* npss */

#define INCL_DOSNMPIPES

USHORT rc = DosQNmPipeSemState(SemHandle, InfoBuf, InfoBufLen);

HSEM      SemHandle;          /* Semaphore handle */
PBYTE     InfoBuf;           /* Address of returned info */
USHORT    InfoBufLen;        /* Length of InfoBuf */

USHORT    rc;                /* return code */
```

### Assembler Language

```
EXTRN DosQNmPipeSemState:FAR
INCL_DOSNMPIPES EQU 1

PUSH  DWORD  SemHandle      ;Semaphore handle
PUSH@ OTHER  InfoBuf        ;Information (returned)
PUSH  WORD   InfoBufLen     ;Length of InfoBuf
CALL  DosQNmPipeSemState

Returns WORD
```

# DosQPathInfo —

## Query File or Subdirectory for Information

DosQPathInfo returns attribute and extended attribute information for a file or subdirectory.

**DosQPathInfo** (**PathName**, **PathInfoLevel**, **PathInfoBuf**, **PathInfoBufSize**, **Reserved**)

### Parameters

**PathName** (*PSZ*) — input

Address of the ASCIIZ full path name of the file or subdirectory. Global file name characters can be used in the name only for PathInfoLevels five and six.

DosQSysInfo is called by an application during initialization to determine the maximum path length allowed by OS/2.

**PathInfoLevel** (*USHORT*) — input

Level of path information required. A value of 1, 2, 3, 5, or 6 can be specified. Level 4 is reserved. The structures described in PathInfoBuf indicate the information returned for each of these levels.

**PathInfoBuf** (*PBYTE*) — output

Address of the storage area containing the requested level of path information. Path information, where applicable, is based on the most recent DosClose, DosBufReset, DosSetFileInfo, or DosSetPathInfo.

#### Level 1 Information

PathInfoBuf contains the following structure, to which path information is returned:

**filedate** (*FDATE*)

Structure containing the date of creation.

Bit	Description
15 – 9	Year, in binary, of creation
8 – 5	Month, in binary, of creation
4 – 0	Day, in binary, of creation.

**filetime** (*FTIME*)

Structure containing the time of creation.

Bit	Description
15 – 11	Hours, in binary, of creation
10 – 5	Minutes, in binary, of creation
4 – 0	Seconds, in binary number of two-second increments, of creation.

**fileaccessdate** (*FDATE*)

Structure containing the date of last access. See *FDATE* in filedate.

**fileaccesstime** (*FTIME*)

Structure containing the time of last access. See *FTIME* in filetime.

**writeaccessdate** (*FDATE*)

Structure containing the date of last write. See *FDATE* in filedate.

**writeaccesstime** (*FTIME*)

Structure containing the time of last write. See *FTIME* in filetime.

**filesize** (*ULONG*)

File size.

**filealloc** (*ULONG*)

Allocated file size.

**fileattrib** (*USHORT*)

Attributes of the file, defined in DosSetFileMode.

# DosQPathInfo —

## Query File or Subdirectory for Information

FAPI

### Level 2 Information

PathInfoBuf contains a structure similar to the Level 1 structure, with the addition of the cbList field after the fileattrib field.

#### cbList (ULONG)

On output, this field contains the length of the entire EA set for the file object. This value can be used to calculate the size of the buffer required to hold EA information returned when PathInfoLevel = 3 is specified.

### Level 3 Information

PathInfoBuf contains an EAOP structure, which has the following format:

#### fpGEAList (PGEALIST)

Address of GEAList. GEAList is a packed array of variable length "get EA" structures, each containing an EA name and the length of the name.

#### fpFEAList (PFEALIST)

Address of FEAList. FEAList is a packed array of variable length "full EA" structures, each containing an EA name and its corresponding value, as well as the lengths of the name and the value.

#### oError (ULONG)

Offset into structure where error has occurred.

On input, PathInfoBuf is an EAOP structure. fpGEAList points to a GEA list defining the attribute names whose values are returned. fpFEAList points to a data area where the relevant FEA list is returned. The length field of this FEA list is valid, giving the size of the FEA list buffer. oError points to the offending GEA entry in case of error. Following is the format of the GEAList structure:

#### cbList (ULONG)

Length of the GEA list, including the length itself.

#### list (GEA)

List of GEA structures. A GEA structure has the following format:

##### cbName (BYTE)

Length of EA ASCIIZ name, which does not include the null character.

##### szName (CHAR)

ASCIIZ name of EA.

On output, PathInfoBuf is unchanged. The buffer pointed to by fpFEAList is filled in with the returned information. If the buffer fpFEAList points to isn't large enough to hold the returned information (ERROR\_BUFFER\_OVERFLOW) cbList is still valid, assuming there's at least enough space for it. Its value is the size of the entire EA set for the file, even though only a subset of attributes was requested. Following is the format of the FEAList structure:

#### cbList (ULONG)

Length of the FEA list, including the length itself.

#### list (FEA)

List of FEA structures. An FEA structure has the following format:

##### Flags (BYTE)

Bit indicator describing the characteristics of the EA being defined.

Bit	Description
-----	-------------

15	Critical EA.
----	--------------

14 – 0	Reserved and must be set to zero.
--------	-----------------------------------

If bit 15 is set to 1, this indicates a critical EA. If bit 15 is 0, this means the EA is noncritical; that is, the EA is not essential to the intended use by an application of the file with which it is associated.

##### cbName (BYTE)

Length of EA ASCIIZ name, which does not include the null character.

## DosQPathInfo —

# Query File or Subdirectory for Information

**cbValue (USHORT)**

Length of EA value, which cannot exceed 64KB.

**szName (PSZ)**

Address of the ASCIIZ name of EA.

**aValue (PSZ)**

Address of the free-format value of EA.

**Note:** The szName and aValue fields are not included as part of header or include files. Because of their variable lengths, these entries must be built manually.

**Level 5 Information**

Level 5 returns the fully qualified ASCIIZ name of PathName in PathInfoBuf. The PathName may contain global file name characters.

**Level 6 Information**

Level 6 requests a file system to verify the correctness of PathName per its rules of syntax. An erroneous name is indicated by an error return code. The PathName may contain global file name characters.

**PathInfoBufSize (USHORT) — output**

Length of PathInfoBuf.

**Reserved (ULONG) — input**

Reserved, must be set to zero.

**rc (USHORT) — return**

Return code descriptions are:

<b>0</b>	<b>NO_ERROR</b>
<b>3</b>	<b>ERROR_PATH_NOT_FOUND</b>
<b>32</b>	<b>ERROR_SHARING_VIOLATION</b>
<b>111</b>	<b>ERROR_BUFFER_OVERFLOW</b>
<b>124</b>	<b>ERROR_INVALID_LEVEL</b>
<b>206</b>	<b>ERROR_FILENAME_EXCED_RANGE</b>
<b>254</b>	<b>ERROR_INVALID_EA_NAME</b>
<b>255</b>	<b>ERROR_EA_LIST_INCONSISTENT</b>

**Remarks**

For DosQPathInfo to return information contained in any of the file information levels, the file object must be opened for read access, with a deny-write sharing mode specified for access by other processes. Thus, if the file object is already accessed by another process that holds conflicting sharing and access rights, a call to DosQPathInfo fails.

**C Language**

```
typedef struct _FDATE {    /* fdate */

    unsigned day   : 5;    /* binary day for directory entry */
    unsigned month : 4;    /* binary month for directory entry */
    unsigned year  : 7;    /* binary year for directory entry */

} FDATE;

typedef struct _FTIME {    /* ftime */

    unsigned twosecs : 5;    /* binary number of two-second increments */
    unsigned minutes : 6;    /* binary number of minutes */
    unsigned hours   : 5;    /* binary number of hours */

} FTIME;
```



## Query File or Subdirectory for Information

```

typedef struct _FILESTATUS {    /* fsts */

    FDATE  fdateCreation;        /* date of file creation */
    FTIME  ftimeCreation;        /* time of file creation */
    FDATE  fdateLastAccess;      /* date of last access */
    FTIME  ftimeLastAccess;      /* time of last access */
    FDATE  fdateLastWrite;       /* date of last write */
    FTIME  ftimeLastWrite;       /* time of last write */
    ULONG  cbFile;               /* file size (end of data) */
    ULONG  cbFileAlloc;          /* file allocated size */
    USHORT attrFile;             /* attributes of the file */

} FILESTATUS;

typedef struct _GEA {           /* gea */

    BYTE  cbName;                /* name length not including NULL */
    CHAR  szName[1];             /* attribute name */

} GEA;

typedef struct _GEALIST {      /* geal */

    ULONG  cbList;               /* total bytes of structure including full list */
    GEA  list[1];               /* variable length GEA structures */

} GEALIST;

typedef struct _FEA {          /* fea */

    BYTE  fEA;                  /* flags */
    BYTE  cbName;               /* name length not including NULL */
    USHORT cbValue;             /* value length */

} FEA;

typedef struct _FEALIST {      /* feal */

    ULONG  cbList;               /* total bytes of structure including full list */
    FEA  list[1];               /* variable length FEA structures */

} FEALIST;

typedef struct _EAOP {         /* eaop */

    PGEALIST fpGEAList;          /* general EA list */
    PFEALIST fpFEAList;          /* full EA list */
    ULONG  oError;

} EAOP;

#define INCL_DOSFILEMGR

USHORT  rc = DosQPathInfo(PathName, PathInfoLevel, PathInfoBuf, PathInfoBufSize, 0);

PSZ      PathName;              /* File or directory path name string */
USHORT   PathInfoLevel;         /* Data required */
PBYTE    PathInfoBuf;           /* Data buffer (returned) */
USHORT   PathInfoBufSize;       /* Data buffer size */
ULONG    0;                    /* Reserved (must be zero) */

USHORT   rc;                   /* return code */

```

## DosQPathInfo — Query File or Subdirectory for Information

### Assembler Language

```
FDATE    struc

    fdate_fs    dw    ?

FDATE    ends

FTIME    struc

    ftime_fs    dw    ?

FTIME    ends

FILESTATUS struc

    fsts_fdateCreation    dw    (size FDATE)/2 dup (?) ;date of file creation
    fsts_ftimeCreation    dw    (size FTIME)/2 dup (?) ;time of file creation
    fsts_fdateLastAccess  dw    (size FDATE)/2 dup (?) ;date of last access
    fsts_ftimeLastAccess  dw    (size FTIME)/2 dup (?) ;time of last access
    fsts_fdateLastWrite   dw    (size FDATE)/2 dup (?) ;date of last write
    fsts_ftimeLastWrite   dw    (size FTIME)/2 dup (?) ;time of last write
    fsts_cbFile           dd    ? ;file size (end of data)
    fsts_cbFileAlloc      dd    ? ;file allocated size
    fsts_attrFile         dw    ? ;attributes of the file

FILESTATUS ends
```

# DosQPathInfo — Query File or Subdirectory for Information

FAPI

```

GEA    struc

    gea_cbName    db    ?           ;name length not including NULL
    gea_szName    db    1 dup (?)   ;attribute name

GEA    ends

GEALIST    struc

    geal_cbList    dd    ?           ;total bytes of structure including full list
    geal_list      db    size GEA * 1 dup (?) ;variable length GEA structures

GEALIST    ends

FEA    struc

    fea_fEA        db    ? ;flags
    fea_cbName      db    ? ;name length not including NULL
    fea_cbValue     dw    ? ;value length

FEA    ends

FEALIST    struc

    feal_cbList     dd    ?           ;total bytes of structure including full list
    feal_list       db    size FEA * 1 dup (?) ;variable length FEA structures

FEALIST    ends

EAOP     struc

    eaop_fpGEAList  dd    ? ;general EA list
    eaop_fpFEAList  dd    ? ;full EA list
    eaop_oError     dd    ? ;

EAOP     ends

EXTRN DosQPathInfo:FAR
INCL_DOSFILEMGR     EQU 1

PUSH@    ASCIIZ PathName           ;File or directory path name string
PUSH     WORD    PathInfoLevel      ;Data required
PUSH@    OTHER   PathInfoBuf        ;Data buffer (returned)
PUSH     WORD    PathInfoBufSize    ;Data buffer size
PUSH     DWORD   0                  ;Reserved (must be zero)
CALL     DosQPathInfo

Returns WORD

```

# DosQSysInfo – Query System Variable Information

This call returns values of static system variables.

**DosQSysInfo (Index, DataBuf, DataBufLen)**

## Parameters

**Index** (*USHORT*) – input

Ordinal of the system variable to return.

Index = 0 indicates maximum path length. The maximum path length is returned in the first word of the DataBuf.

**DataBuf** (*PBYTE*) – output

Address where the system returns the variable value.

**DataBufLen** (*USHORT*) – input

Length of the data buffer.

**rc** (*USHORT*) – return

Return code descriptions include:

0	NO_ERROR
87	ERROR_INVALID_PARAMETER
111	ERROR_BUFFER_OVERFLOW

## Remarks

An OS/2 application may want to reference file objects managed by an installable file system that supports long file names. Because some installable file systems may support longer names than others, an application should issue DosQSysInfo upon initialization.

DosQSysInfo returns the maximum path length supported by the file system currently installed. The path length includes the drive specifier (d:), the leading "\" and the trailing null character.

The value returned by DosQSysInfo can be used to allocate buffers for storing path names returned by requests, for example, to DosFindFirst and DosFindNext.

## C Language

```
#define INCL_DOSFILEMGR
```

```
USHORT rc = DosQSysInfo(Index, DataBuf, DataBufLen);
```

```
USHORT      Index;      /* Which variable */
PBYTE       DataBuf;    /* System information (returned) */
USHORT      DataBufLen; /* Data buffer size */

USHORT      rc;          /* return code */
```

## Assembler Language

```
EXTRN DosQSysInfo:FAR
INCL_DOSFILEMGR EQU 1
```

```
PUSH WORD Index      ;Which variable
PUSH@ OTHER DataBuf   ;System information (returned)
PUSH WORD DataBufLen  ;Data buffer size
CALL DosQSysInfo
```

Returns WORD

# DosQueryQueue — Query Size of Queue

---

This call determines the number of elements in a queue.

**DosQueryQueue (QueueHandle, NumberElements)**

## Parameters

**QueueHandle** (*HQUEUE*) — input

Handle of the queue to find size.

**NumberElements** (*PUSHORT*) — output

Address of the number of entries in the queue waiting to be processed.

**rc** (*USHORT*) — return

Return code descriptions are:

0	NO_ERROR
337	ERROR_QUE_INVALID_HANDLE

## Remarks

Any thread of a process that has access to the queue (because of a `DosCreateQueue` or a `DosOpenQueue` request) may issue `DosQueryQueue` to determine the number of elements currently in the queue.

If the process that created the queue closes it before this request is issued, `ERROR_QUE_INVALID_HANDLE` is returned.

## C Language

```
#define INCL_DOSQUEUES

USHORT rc = DosQueryQueue(QueueHandle, NumberElements);

HQUEUE QueueHandle; /* Queue handle */
PUSHORT NumberElements; /* Size of the queue (returned) */

USHORT rc; /* return code */
```

## Assembler Language

```
EXTRN DosQueryQueue:FAR
INCL_DOSQUEUES EQU 1

PUSH WORD QueueHandle ;Queue handle
PUSH@ WORD NumberElements ;Size of the queue (returned)
CALL DosQueryQueue

Returns WORD
```

---

This call returns the value of the verify flag.

<b>DosQVerify (VerifySetting)</b>
-----------------------------------

## Parameters

**VerifySetting** (*PUSHORT*) – output

Address of the verify mode for the process.

Value	Definition
00H	Verify mode is not active.
01H	Verify mode is active.

**rc** (*USHORT*) – return

Return code description is:

0	NO_ERROR
---	----------

## Remarks

When verify is active, OS/2 performs a verify operation each time it does a file write to assure proper data recording on the disk. Although disk recording errors are rare, this function has been provided for applications to verify the proper recording of critical data.

## C Language

```
#define INCL_DOSFILEMGR

USHORT rc = DosQVerify(VerifySetting);

PUSHORT VerifySetting; /* Verify setting (returned) */

USHORT rc; /* return code */
```

## Assembler Language

```
EXTRN DosQVerify:FAR
INCL_DOSFILEMGR EQU 1

PUSH@ WORD VerifySetting ;Verify setting (returned)
CALL DosQVerify

Returns WORD
```

# DosR2StackRealloc – Reallocate Privilege Level 2 Stack

---

This call changes the size of a thread's privilege level 2 stack.

**DosR2StackRealloc (NewSize)**

## Parameters

**NewSize** (*USHORT*) – input

New size, in bytes, for the privilege level 2 stack. The stack is reallocated as required and the TSS SP pointer is adjusted accordingly. The new stack size can not be less than the current stack size.

**rc** (*USHORT*) – return

Return code descriptions are:

<b>0</b>	<b>NO_ERROR</b>
<b>8</b>	<b>ERROR_NOT_ENOUGH_MEMORY</b>
<b>196</b>	<b>ERROR_DYNLINK_FROM_INVALID_RING</b>
<b>197</b>	<b>ERROR_IOPL_NOT_ENABLED</b>
<b>207</b>	<b>ERROR_RING2_STACK_IN_USE</b>
<b>216</b>	<b>ERROR_CANNOT_SHRINK</b>

## Remarks

This call is provided to allow the privilege level 2 stack to be resized and to have the TSS adjusted to reflect the new size. The size can not be less than the current size.

This call can not be made from privilege level 2.

## C Language

```
#define INCL_DOSDEVICES
```

```
USHORT rc = DosR2StackRealloc(NewSize);
```

```
USHORT      NewSize;      /* The new size in bytes for the stack */
```

```
USHORT      rc;           /* return code */
```

## Assembler Language

```
EXTRN DosR2StackRealloc:FAR  
INCL_DOSDEVICES EQU 1
```

```
PUSH WORD NewSize ;The new size in bytes for the stack  
CALL DosR2StackRealloc
```

Returns WORD

This call reads the specified number of bytes from a file, pipe, or device to a buffer location.

**DosRead** (*FileHandle*, *BufferArea*, *BufferLength*, *BytesRead*)

## Parameters

**FileHandle** (*HFILE*) — input

File handle obtained from DosOpen.

**BufferArea** (*PVOID*) — output

Address of the input buffer.

**BufferLength** (*USHORT*) — input

Length, in bytes, to be read.

**BytesRead** (*PUSHORT*) — output

Address of the number of bytes read.

**rc** (*USHORT*) — return

Return code descriptions are:

0	NO_ERROR
5	ERROR_ACCESS_DENIED
6	ERROR_INVALID_HANDLE
26	ERROR_NOT_DOS_DISK
33	ERROR_LOCK_VIOLATION
109	ERROR_BROKEN_PIPE
234	ERROR_MORE_DATA

## Remarks

The requested number of bytes may not be read. If the value returned in BytesRead is zero, the process has tried to read from the end of the file.

A BufferLength value of zero is not considered an error. In the case where BufferLength is zero, the system treats the request as a null operation.

The file pointer is moved to the desired position by reading, writing, or issuing DosChgFilePtr.

## Family API Considerations

Some options operate differently in the DOS mode than in the OS/2 mode. Therefore, the following restrictions apply to DosRead when coding for the DOS mode:

- Only single-byte DosRead calls may be made to the COM device, because the COM device driver for the DOS environment does not support multiple-byte I/O.
- When DosRead is called with a handle that is open to CON, the read goes directly through KbdStringIn using the buffer and length that are provided to DosRead. This causes a DOS mode DosRead to behave differently than an OS/2 mode DosRead. Because an OS/2 mode DosRead buffers the call to KbdStringIn, this allows the user to enter many more characters.

For example, suppose DosRead is called with a buffer of length 10 from a handle opened to CON:

- In OS/2 mode, the user is allowed to enter a large number of characters before KbdStringIn begins beeping (indicating the buffer is full). After carriage return is pressed, KbdStringIn returns to DosRead. DosRead returns the first 10 characters to the caller and the remaining characters on subsequent calls to DosRead from CON.
- In DOS mode, the user is allowed to enter only eight characters (DOS mode DosRead reserves two characters for CR and LF) before KbdStringIn begins beeping. DosRead returns the eight characters entered, followed by CR and LF to the calling program.



### Named Pipe Considerations

A named pipe is read as one of the following:

- A byte pipe in byte read mode
- A message pipe in message read mode
- A message pipe in byte read mode.

A **byte pipe** must be in byte read mode to be read; an error is returned if it is in message read mode. All currently available data, up to the size requested, is returned.

A **message pipe** can be read in either message read mode or byte read mode. When the message pipe is in message read mode, a read that is larger than the next available message returns only that message. BytesRead is set to indicate the size of the message returned.

A read that is smaller than the next available message returns with the number of bytes requested and an ERROR\_MORE\_DATA return code. When resuming the reading of a message after ERROR\_MORE\_DATA is returned, a read always blocks until the next piece (or the rest) of the message can be transferred. DosPeekNmPipe may be used to determine how many bytes are left in the message.

A message pipe in byte read mode is read as if it were a byte stream, skipping over message headers. This is like reading a byte pipe in byte read mode.

When blocking mode is set for a named pipe, a read blocks until data is available. In this case, the read never returns with BytesRead = 0 except at EOF. In message read mode, messages are always read in their entirety, except in the case where the message is bigger than the size of the read.

Non-blocking mode allows a return with BytesRead = 0 only when no data is available at the time of the read.

### C Language

```
#define INCL_DOSFILEMGR

USHORT rc = DosRead(FileHandle, BufferArea, BufferLength, BytesRead);

HFILE      FileHandle;    /* File Handle */
PVOID      BufferArea;     /* User buffer (returned) */
USHORT     BufferLength;   /* Buffer length */
PUSHORT    BytesRead;     /* Bytes read (returned) */

USHORT     rc;            /* return code */
```

### Assembler Language

```
EXTRN DosRead:FAR
INCL_DOSFILEMGR EQU 1

PUSH WORD FileHandle ;File Handle
PUSH@ OTHER BufferArea ;User buffer (returned)
PUSH WORD BufferLength ;Buffer length
PUSH@ WORD BytesRead ;Bytes read (returned)
CALL DosRead

Returns WORD
```

# DosReadAsync – Asynchronous Read from File

This call asynchronously transfers the specified number of bytes from a file, pipe, or device to a buffer.

**DosReadAsync** (*FileHandle*, *RamSemaphore*, *ReturnCode*, *BufferArea*,  
*BufferLength*, *BytesRead*)

## Parameters

**FileHandle** (*HFILE*) – input

File handle obtained from DosOpen.

**RamSemaphore** (*PULONG*) – input

Address used by the system to signal the caller that the read operation is complete.

**ReturnCode** (*PUSHORT*) – output

Address of the I/O error return code.

**BufferArea** (*PVOID*) – output

Address of the input buffer.

**BufferLength** (*USHORT*) – input

Length, in bytes, to be read.

**BytesRead** (*PUSHORT*) – output

Address of the number of bytes read.

**rc** (*USHORT*) – return

Return code descriptions are:

0	NO_ERROR
5	ERROR_ACCESS_DENIED
6	ERROR_INVALID_HANDLE
26	ERROR_NOT_DOS_DISK
33	ERROR_LOCK_VIOLATION
89	ERROR_NO_PROC_SLOTS
109	ERROR_BROKEN_PIPE

## Remarks

The requested number of bytes may not be read. If the value in BytesRead is zero, the process tried to read from the end of the file.

A BufferLength value of zero is not considered an error. In the case where BufferLength is zero, the system treats the request as a null operation.

The file pointer is moved to the desired position by reading, writing, or issuing DosChgFilePtr.

A call to DosReadAsync may return before the read is complete. To wait for an asynchronous read to complete, RamSemaphore must be set by the application before the DosReadAsync call is made. The application issues DosSemSet to set the semaphore, calls DosReadAsync, and then issues DosSemWait, to wait to be signaled by the system that the asynchronous read is complete. When RamSemaphore is cleared and the read operation is complete, ReturnCode can be checked.

**Note:** If it is necessary to make a DosReadAsync request from within a segment that has I/O privilege, DosCallback may be used to invoke a privilege level 3 segment that actually issues the DosReadAsync request.

# DosReadAsync — Asynchronous Read from File

## Named Pipe Considerations

A named pipe is read as one of the following:

- A byte pipe in byte read mode
- A message pipe in message read mode
- A message pipe in byte read mode.

A **byte pipe** must be in byte read mode to be read; an error is returned if it is in message read mode. All currently available data, up to the size requested, is returned.

When a **message pipe** is read in message read mode, a read that is larger than the next available message returns only that message and BytesRead is set to indicate the size of the message returned.

A read that is smaller than the next available message returns with the number of bytes requested and an ERROR\_MORE\_DATA return code. When resuming the reading of a message after ERROR\_MORE\_DATA is returned, a read always blocks until the next piece (or rest) of the message can be transferred. DosPeekNmPipe may be used to determine how many bytes are left in the message.

A message pipe in byte read mode is read as if it were a byte stream, skipping over message headers. This is like reading a byte pipe in byte read mode.

When blocking mode is set for a named pipe, a read blocks until data is available. In this case, the read never returns with BytesRead = 0 except at EOF. In message read mode, messages are always read in their entirety, except in the case where the message is bigger than the size of the read.

Non-blocking mode allows a return with BytesRead = 0 only when trying to read at the start of a message and no message is available.

## C Language

```
#define INCL_DOSFILEMGR
```

```
USHORT rc = DosReadAsync(FileHandle, RamSemaphore, ReturnCode, BufferArea,  
                          BufferLength, BytesRead);
```

```
HFILE      FileHandle;    /* File handle */  
PULONG     RamSemaphore;  /* Ram semaphore */  
PUSHORT    ReturnCode;    /* I/O operation return code (returned) */  
PVOID      BufferArea;    /* User buffer (returned) */  
USHORT     BufferLength;  /* Buffer length */  
PUSHORT    BytesRead;     /* Bytes read (returned) */  
  
USHORT     rc;            /* return code */
```

## Assembler Language

```
EXTRN DosReadAsync:FAR  
INCL_DOSFILEMGR EQU 1
```

```
PUSH WORD FileHandle ;File handle  
PUSH@ DWORD RamSemaphore ;Ram semaphore  
PUSH@ WORD ReturnCode ;I/O operation return code (returned)  
PUSH@ OTHER BufferArea ;User buffer (returned)  
PUSH WORD BufferLength ;Buffer length  
PUSH@ WORD BytesRead ;Bytes read (returned)  
CALL DosReadAsync
```

Returns WORD

## DosReadQueue — Read from Queue

This call reads an element from a queue and removes it.

**DosReadQueue** (*QueueHandle*, *Request*, *DataLength*, *DataAddress*, *ElementCode*, *NoWait*, *ElemPriority*, *SemaphoreHandle*)

### Parameters

**QueueHandle** (*HQUEUE*) — input

Handle of the queue to read from.

**Request** (*PULONG*) — output

Address of a data field that returns the following information.

The first word is the PID of the process that added the element to the queue.

The second word is used for event encoding by the application. The data in this word is the same as that furnished by the Request parameter on the DosWriteQueue request for the corresponding queue element.

**DataLength** (*PUSHORT*) — output

Address of the length of the data being received.

**DataAddress** (*PULONG*) — output

Address of the element being received from the queue.

**ElementCode** (*USHORT*) — input

Overrides the normal priority, FIFO-, or LIFO-read ordering. This operand is used to identify a specific element that is to be read. This field can be set to zero to read the first element in the queue, or it can contain an identifier for a particular element, which was returned to ElementCode by a DosPeekQueue request.

**NoWait** (*UCHAR*) — input

Action to be performed when no entries are found in the queue.

Value	Definition
0	The requesting thread waits.
1	The requesting thread does not wait.

**ElemPriority** (*PBYTE*) — output

Address of the element's priority. This is the value that was specified for ElemPriority by the DosWriteQueue call that added the element to the queue. ElemPriority is a numeric value in the range of 0 to 15, with 15 being the highest priority.

**SemaphoreHandle** (*HSEM*) — input

Handle of the semaphore cleared when an element is written to the queue and NoWait=0 is specified. If NoWait=1 is specified, this parameter should be set to null.

The semaphore can be either a RAM or system semaphore. If the semaphore is a RAM semaphore, it must be in a segment that is shared between the process that owns the queue and any process that issues a DosWriteQueue request to the queue.

If multiple threads are processing elements from the queue using NoWait=0, the same semaphore must be provided on all DosPeekQueue or DosReadQueue requests.

**rc** (*USHORT*) — return

Return code descriptions are:

0	NO_ERROR
330	ERROR_QUE_PROC_NOT_OWNED
333	ERROR_QUE_ELEMENT_NOT_EXIST
337	ERROR_QUE_INVALID_HANDLE
342	ERROR_QUE_EMPTY
433	ERROR_QUE_INVALID_WAIT

# DosReadQueue —

## Read from Queue

### Remarks

A process that creates a queue with `DosCreateQueue` owns it. Only the owning process and any threads it creates can issue `DosReadQueue` to remove an element from the queue. If the queue is empty and `NoWait = 0` is specified, the thread waits for an element to be written to the queue. If the queue is empty and `NoWait = 1` is specified, the thread returns with `ERROR_QUEUE_EMPTY`.

If `ElementCode` is set to zero, the elements in the queue are removed in the order specified at the time of the queue's creation (FIFO, LIFO, or priority).

`ElementCode` can also be set to a value returned by `DosPeekQueue`, which uses `ElementCode` to return identifiers for successive queue elements. The assigning of identifiers by `DosPeekQueue` to individual queue elements allows the queue owner to examine a queue element with `DosPeekQueue` and compare it with a queue element it has read.

The semaphore provided by `SemaphoreHandle` is typically used by a `DosMuxSemWait` request to wait for an element to be written to a queue or other events. If `DosReadQueue` is issued with `NoWait=0`, it clears the semaphore indicated by `SemaphoreHandle` as soon as the element is read.

The `Request`, `DataLength` and `DataAddress` parameters contain data understood by the thread adding the element to the queue and by the thread that receives the queue element. There is no special meaning to this data; applications may use these parameters for any purpose they wish. OS/2 does not alter this data; it simply copies this data intact. OS/2 does not validate the address of `DataBuffer` or the `DataLength`.

### C Language

```
#define INCL_DOSQUEUES
```

```
USHORT rc = DosReadQueue(QueueHandle, Request, DataLength, DataAddress,  
                          ElementCode, NoWait, ElemPriority, SemaphoreHandle);
```

```
HQUEUE      QueueHandle;    /* Queue handle */  
PULONG      Request;        /* Request identification data (returned) */  
PUSHORT      DataLength;     /* Length of element received (returned) */  
PULONG      DataAddress;     /* Address of element received (returned) */  
USHORT      ElementCode;    /* Indicate want a particular element */  
UCHAR       NoWait;         /* Indicate to not wait if queue is empty */  
PBYTE       ElemPriority;    /* Priority of element (returned) */  
HSEM        SemaphoreHandle; /* Semaphore handle */
```

```
USHORT      rc;             /* return code */
```

### Assembler Language

```
EXTRN DosReadQueue:FAR  
INCL_DOSQUEUES EQU 1
```

```
PUSH WORD QueueHandle ;Queue handle  
PUSH@ DWORD Request ;Request identification data (returned)  
PUSH@ WORD DataLength ;Length of element received (returned)  
PUSH@ DWORD DataAddress ;Address of element received (returned)  
PUSH WORD ElementCode ;Indicate want a particular element  
PUSH OTHER NoWait ;Indicate to not wait if queue is empty  
PUSH@ BYTE ElemPriority ;Priority of element (returned)  
PUSH DWORD SemaphoreHandle ;Semaphore handle  
CALL DosReadQueue
```

Returns WORD

This call changes the size of memory originally allocated by DosAllocHuge.

**DosReallocHuge (NumSeg, Size, Selector)**

## Parameters

**NumSeg** (*USHORT*) — input

Number of 65536 byte segments requested.

**Size** (*USHORT*) — input

Number of bytes requested in the last non-65536 byte segment. A value of 0 indicates none.

**Selector** (*SEL*) — input

Selector returned on a previous DosAllocHuge.

**rc** (*USHORT*) — return

Return code descriptions are:

0	NO_ERROR
8	ERROR_NOT_ENOUGH_MEMORY
87	ERROR_INVALID_PARAMETER

## Remarks

DosReallocHuge is called to change the size of unshared or shared huge memory allocated by DosAllocHuge. The selector used for this call must be the one returned by the DosAllocHuge request.

Normally, segments allocated as shared (AllocFlags bits 0 and 1 were set) cannot be decreased in size. However, if AllocFlags bit 3 was also set, the shared segment's size can be decreased.

DosReallocHuge is also called to reallocate a segment allocated as discardable (AllocFlags bit 2 set) after the segment is discarded by the system. The call to DosReallocHuge automatically locks the segment for access by the caller, the same as if a DosLockSeg had been issued.

**Note:** This request may be issued from privilege level 2 or 3. However, only a privilege level 3 huge segment is valid.

## Family API Considerations

Some options operate differently in the DOS mode than in the OS/2 mode. Therefore, the following restriction applies to DosReallocHuge when coding for the DOS mode:

The requested Size value is rounded up to the next paragraph (16-byte).

## C Language

```
#define INCL_DOSMEMMGR
```

```
USHORT rc = DosReallocHuge(NumSeg, Size, Selector);
```

```
USHORT    NumSeg;    /* Number of 65536-byte segments requested */
USHORT    Size;      /* Number of bytes in last segment */
SEL       Selector;  /* Selector */

USHORT    rc;        /* return code */
```

# DosReallocHuge — Change Huge Memory Size

FAPI

## Assembler Language

```
EXTRN  DosReallocHuge:FAR
INCL_DOSMEMMGR      EQU 1

PUSH  WORD    NumSeg      ;Number of 65536-byte segments requested
PUSH  WORD    Size       ;Number of bytes in last segment
PUSH  WORD    Selector    ;Selector
CALL  DosReallocHuge

Returns WORD
```

---

This call reallocates a segment after discard or changes the size of a segment already allocated.

<b>DosReallocSeg (Size, Selector)</b>
---------------------------------------

## Parameters

**Size** (*USHORT*) — input

New requested segment size (in bytes). A value of 0 indicates 65536 bytes.

**Selector** (*SEL*) — input

Segment to be resized.

**rc** (*USHORT*) — return

Return code descriptions are:

<b>0</b>	<b>NO_ERROR</b>
<b>5</b>	<b>ERROR_ACCESS_DENIED</b>
<b>8</b>	<b>ERROR_NOT_ENOUGH_MEMORY</b>

## Remarks

DosReallocSeg is called to change the size of an unshared or shared segment allocated with a DosAllocSeg request.

Normally, segments allocated as shared (AllocFlags bits 0 and 1 were set) cannot be decreased in size. However, if AllocFlags bit 3 was also set, the shared segment's size can be decreased.

DosReallocSeg is also called to reallocate a segment allocated as discardable (AllocFlags bit 2 set) after the segment is discarded by the system. The call to DosReallocSeg automatically locks the segment for access by the caller, the same as if a DosLockSeg had been issued.

**Note:** This request may be issued from privilege level 2 or 3, and the segment being resized can be either a privilege level 2 or privilege level 3 segment.

To change the size of huge memory allocated by DosAllocHuge, see DosReallocHuge.

## Family API Considerations

Some options operate differently in the DOS mode than in the OS/2 mode. Therefore, the following restriction applies to DosReallocSeg when coding for the DOS mode. The requested Size value is rounded up to the next paragraph (16-byte).

## C Language

```
#define INCL_DOSMEMMGR
```

```
USHORT rc = DosReallocSeg(Size, Selector);
```

```
USHORT      Size;          /* New size requested in bytes */
SEL          Selector;     /* Selector */
```

```
USHORT      rc;            /* return code */
```



# DosReallocSeg — Change Segment Size

FAP1

## Assembler Language

```
EXTRN DosReallocSeg:FAR
INCL_DOSMEMMGR      EQU 1

PUSH  WORD    Size          ;New size requested in bytes
PUSH  WORD    Selector      ;Selector
CALL  DosReallocSeg
```

Returns WORD

# DosResumeThread – Restart Thread

---

This call restarts a thread previously stopped with a DosSuspendThread call.

<b>DosResumeThread (ThreadID)</b>
-----------------------------------

## Parameters

**ThreadID (TID)** – input  
Thread ID of the resumed thread.

**rc (USHORT)** – return  
Return code descriptions are:

<b>0</b>	<b>NO_ERROR</b>
<b>309</b>	<b>ERROR_INVALID_THREADID</b>

## C Language

```
#define INCL_DOSPROCESS

USHORT rc = DosResumeThread(ThreadID);

TID      ThreadID;      /* Thread ID of thread to resume */

USHORT    rc;            /* return code */
```

## Assembler Language

```
EXTRN DosResumeThread:FAR
INCL_DOSPROCESS EQU 1

PUSH WORD ThreadID ;Thread ID
CALL DosResumeThread

Returns WORD
```

## Example

The following example shows how to suspend and resume execution of a thread within a process. The main thread creates Thread2 and allows it to begin executing. Thread2 iterates through a loop that prints a line and then sleeps, relinquishing its time slice to the main thread. After one iteration by Thread2, the main thread suspends Thread2 and then resumes it. Subsequently, Thread2 completes the remaining three iterations.

## DosResumeThread — Restart Thread

```
#define INCL_DOSPROCESS

#include <os2.h>

#define SEGSIZE      4000    /* Number of bytes requested in segment */
#define ALLOCFLAGS    0      /* Segment allocation flags - no sharing */
#define SLEEPSHORT    5L     /* Sleep interval - 5 milliseconds */
#define SLEEPLONG     75L    /* Sleep interval - 75 milliseconds */
#define RETURN_CODE    0      /* Return code for DosExit() */

VOID APIENTRY Thread2()
{
    USHORT    i;

    /* Loop with four iterations */
    for(i=1; i<5; i++)
    {
        printf("In Thread2, i is now %d\n", i);

        /* Sleep to relinquish time slice to main thread */
        DosSleep(SLEEPSHORT);    /* Sleep interval */
    }
    DosExit(EXIT_THREAD,    /* Action code - end a thread */
            RETURN_CODE);    /* Return code */
}

main()
{
    TID        ThreadID;    /* Thread identification */
    SEL        ThreadStackSel;    /* Segment selector for thread stack */
    PBYTE      StackEnd;    /* Ptr. to end of thread stack */
    USHORT     rc;

    /** Allocate segment for thread stack; make pointer to end of stack. **/
    /** We must allocate a segment in order to preserve segment protection **/
    /** for the thread. **/

    rc = DosAllocSeg(SEGSIZE,    /* Number of bytes requested */
                    &ThreadStackSel,    /* Segment selector (returned) */
                    ALLOCFLAGS);    /* Allocation flags - no sharing */
    StackEnd = MAKEP(ThreadStackSel, SEGSIZE-1);

    /** Start Thread2 **/
    if(!rc=DosCreateThread((PFNTHREAD) Thread2,    /* Thread address */
                        &ThreadID,    /* Thread ID (returned) */
                        StackEnd))    /* End of thread stack */
        printf("Thread2 created.\n");

    /* Sleep to relinquish time slice to Thread2 */
    if(!DosSleep(SLEEPSHORT))    /* Sleep interval */
        printf("Slept a little to let Thread2 execute.\n");

    /***** Suspend Thread2, do some work, then resume Thread2 *****/
    if(!rc=DosSuspendThread(ThreadID))    /* Thread ID */
        printf("Thread2 SUSPENDED.\n");
    printf("Perform work that will not be interrupted by Thread2.\n");
    if(!rc=DosResumeThread(ThreadID))    /* Thread ID */
        printf("Thread2 RESUMED.\n");
    printf("Now we may be interrupted by Thread2.\n");

    /* Sleep to allow Thread2 to complete */
    DosSleep(SLEEPLONG);    /* Sleep interval */
}
```

This call removes a subdirectory from the specified disk.

**DosRmdir** (*DirName*, *Reserved*)

## Parameters

**DirName** (*PSZ*) — input

Address of the fully qualified path name of the subdirectory being removed.

**Reserved** (*ULONG*) — input

Reserved must be set to zero.

**rc** (*USHORT*) — return

Return code descriptions are:

<b>0</b>	<b>NO_ERROR</b>
<b>2</b>	<b>ERROR_FILE_NOT_FOUND</b>
<b>3</b>	<b>ERROR_PATH_NOT_FOUND</b>
<b>5</b>	<b>ERROR_ACCESS_DENIED</b>
<b>16</b>	<b>ERROR_CURRENT_DIRECTORY</b>
<b>26</b>	<b>ERROR_NOT_DOS_DISK</b>
<b>87</b>	<b>ERROR_INVALID_PARAMETER</b>
<b>108</b>	<b>ERROR_DRIVE_LOCKED</b>
<b>208</b>	<b>ERROR_FILENAME_EXCED_RANGE</b>

## Remarks

The subdirectory must be empty, which means it cannot contain hidden files or directory entries other than the "." and ".." entries. Files can be deleted with DosDelete.

The root directory and current directory cannot be removed.

## C Language

```
#define INCL_DOSFILEMGR
```

```
USHORT rc = DosRmdir(DirName, Reserved);
```

```
PSZ      DirName;      /* Directory name string */
ULONG    0;             /* Reserved (must be zero) */
```

```
USHORT    rc;           /* return code */
```

## Assembler Language

```
EXTRN DosRmdir:FAR
INCL_DOSFILEMGR EQU 1
```

```
PUSH@ ASCIIZ DirName      ;Directory name string
PUSH  DWORD 0              ;Reserved (must be zero)
CALL  DosRmdir
```

Returns WORD

# DosScanEnv —

## Scan an Environment Segment

This call scans (searches) an environment segment for an environment variable.

**DosScanEnv (EnvVarName, ResultPointer)**

### Parameters

**EnvVarName (PSZ)** — input

Address of the name of the environment variable. Do not include a trailing "=", since this is not part of the name.

**ResultPointer (PSZ FAR \*)** — output

Address where the system returns the pointer to the environment string. ResultPointer points to the first character of the string that is the value of the environment variable and can be passed directly into DosSearchPath.

**rc (USHORT)** — return

Return code descriptions are:

0	NO_ERROR
203	ERROR_ENVVAR_NOT_FOUND

### Remarks

Assume that the process' environment contains:

"DPATH=c:\sysdir;c:\libdir"

↑  
----- ResultPointer points here after the  
following call to DosScanEnv:

DosScanEnv("DPATH", &ResultPointer);

As noted above, ResultPointer points to the first character of the value of the environment variable.

### C Language

```
#define INCL_DOSQUEUES

USHORT rc = DosScanEnv(EnvVarName, ResultPointer);

PSZ EnvVarName; /* Environment variable name string */
PSZ FAR * ResultPointer; /* Search result pointer (returned) */

USHORT rc; /* return code */
```

### Assembler Language

```
EXTRN DosScanEnv:FAR
INCL_DOSQUEUES EQU 1

PUSH@ ASCIIZ EnvVarName ;Environment variable name string
PUSH@ DWORD ResultPointer ;Search result pointer (returned)
CALL DosScanEnv
```

Returns WORD

## DosScanEnv — Scan an Environment Segment

### Example

The following example scans the environment segment for the PATH variable and prints its value. It then searches the path given by inserting the current directory into the value of the PATH variable for the file named "cmd.exe", and prints the full filename.

```
#define INCL_DOS

#include <os2.h>

#define ENVVARIABLE      "PATH"      /* Environment variable name */
#define FILENAME         "cmd.exe"   /* File for which to search */
#define SEARCH_CUR_DIRECTORY 0x03    /* Search control - current dir., */
                                   /* then environment variable */

main()
{
    PSZ FAR *ResultPointer;          /* Environment scan result pointer (returned) */
    BYTE ResultBuffer[128];          /* Path search result (returned) */
    USHORT rc;                       /* return code */

    /** Scan environment segment for PATH; notice the far-string pointer **/
    /** specification ("%Fs") used to print. **/

    if(!(rc=DosScanEnv(ENVVARIABLE, /* Environment variable name */
                      &ResultPointer)) /* Scan result pointer (returned) */
        printf("%s is %Fs\n", ENVVARIABLE, ResultPointer);

    /** Search current directory + PATH variable for "cmd.exe" **/
    if(!(rc=DosSearchPath(SEARCH_CUR_DIRECTORY, /* Search control vector */
                        ENVVARIABLE, /* Search path reference string */
                        FILENAME, /* File name string */
                        ResultBuffer, /* Search result (returned) */
                        sizeof(ResultBuffer)))) /* Length of search result */
        printf("Found desired file -- %s\n", ResultBuffer);
}
```

# DosSearchPath — Search Path for File Name

This call provides a general path search mechanism that allows applications to find files residing along paths. The path string may come from the process environment, or be supplied directly by the caller.

DosSearchPath (Control, PathRef, FileName, ResultBuffer, ResultBufferLen)

## Parameters

**Control (USHORT)** — input

A word bit vector that controls the behavior of DosSearchPath.

Bit	Description
15 – 3	Reserved bits, must be zero.
2	Ignore network errors bit. The Ignore Network Errors Bit controls whether the search will abort if it encounters a network error or will continue the search with the next element. This allows one to place network paths in the PATH variable and be able to find executables in components of the PATH variable even if the network returns an error, for example, if a server is down. If the Ignore Network Errors Bit = 0, DosSearchPath will abort the search if it encounters an error from the network. If the Ignore Network Errors Bit = 1, DosSearchPath will continue on the search if it encounters network errors.
1	Path source bit. The path source bit determines how DosSearchPath interprets the PathRef argument.  0 = The PathRef points to the actual search path. The search path string may be anywhere in the calling process's address space. Therefore, it may be in the environment, but is not required.  1 = The PathRef points to the name of an environment variable in the process environment, and that environment variable contains the search path.
0	Implied current bit. The implied current bit controls whether the current directory is implicitly on the front of the search path.  0 = DosSearchPath only searches the current directory if it appears in the search path.  1 = DosSearchPath searches the current working directory before it searches the directories in the search path.  For example, implied current bit = 0 and path = ".\;a;b" is equivalent to implied current bit = 1 and path = "a;b".

**PathRef (PSZ)** — input

Address of the path. If the path source bit of control = 0, PathRef is the search path that may be anywhere in the caller's address space.

A search path consists of a sequence of paths separated by a semicolon (;). It is a single ASCIIZ string. The directories are searched in the order they appear in the path.

If the path source bit of control = 1, PathRef is the name of an environment variable, that contains the search path.

A search path consists of a sequence of paths separated by ";". It is a single ASCIIZ string. The directories are searched in the order they appear in the path. Paths that contain ";"s should be quoted. For example:

```
"c:\this is ; one directory path";thisisanother
```

Environment variable names are simply strings that match name strings in the environment. The equal (=) sign is not part of the name.

# DosSearchPath — Search Path for File Name

## **FileName (PSZ) — input**

Address of the ASCIIZ file name. It may contain global file name characters. If FileName does contain global file name characters, they remain in the result path returned in ResultBuffer. This allows applications like CMD.EXE to feed the output directly to DosFindFirst. If there are no global file name characters in FileName, the resulting path returned in ResultBuffer is a full qualified name, and may be passed directly to DosOpen, or any other system call.

## **ResultBuffer (PBYTE) — output**

Address of the pathname of the file, if found. If FileName is found in one of the directories along the path, its full pathname is returned in ResultBuffer (with global file name characters from FileName left in place.) Do not depend on the contents of ResultBuffer being meaningful if DosSearchPath returns a non-zero return code.

## **ResultBufferLen (USHORT) — input**

Length, in bytes, of the ResultBuffer.

## **rc (USHORT) — return**

Return code descriptions are:

<b>0</b>	<b>NO_ERROR</b>
<b>1</b>	<b>ERROR_INVALID_FUNCTION</b>
<b>2</b>	<b>ERROR_FILE_NOT_FOUND</b>
<b>87</b>	<b>ERROR_INVALID_PARAMETER</b>
<b>111</b>	<b>ERROR_BUFFER_OVERFLOW</b>
<b>203</b>	<b>ERROR_ENVVAR_NOT_FOUND</b>

## **Remarks**

PathRef always points to an ASCIIZ string. Let DPATH be an environment variable in the environment segment of the process.

```
"DPATH=c:\sysdir;c:\init" /* in the environment */
```

The following two code fragments are equivalent:

```
DosScanEnv("DPATH", &PathRef);  
DosSearchPath(0, /* Path Source Bit = 0 */  
    PathRef, "myprog.ini", &ResultBuffer, ResultBufLen);
```

```
DosSearchPath(2, /* Path Source Bit = 1 */  
    "DPATH", "myprog.ini", &ResultBuffer, ResultBufLen);
```

They both use the search path stored as DPATH in the environment segment. In the first case, the application uses DosScanEnv to find the variable: in the second case DosSearchPath calls DosScanEnv for the application.

DosSearchPath does not check for consistency or formatting on the names. It does a DosFindFirst on a series of names it constructs from PathRef and FileName.

To determine the size of the returned path name, the ResultBuffer must be scanned for the ASCIIZ terminator.

DosQSysInfo must be used by an application to determine the maximum path length supported by OS/2. The returned value should be used to dynamically allocate buffers that are to be used to store paths.



# DosSearchPath —

## Search Path for File Name

### C Language

```
#define INCL_DOSQUEUES

USHORT rc = DosSearchPath(Control, PathRef, FileName, ResultBuffer,
                          ResultBufferLen);

USHORT      Control;      /* Function control vector */
PSZ          PathRef;     /* Search path reference string */
PSZ          FileName;    /* File name string */
PBYTE        ResultBuffer; /* Search result buffer (returned) */
USHORT       ResultBufferLen; /* Search result buffer length */

USHORT      rc;           /* return code */
```

### Assembler Language

```
EXTRN DosSearchPath:FAR
INCL_DOSQUEUES EQU 1

PUSH WORD Control ;Function control vector
PUSH@ ASCIIZ PathRef ;Search path reference string
PUSH@ ASCIIZ FileName ;File name string
PUSH@ OTHER ResultBuffer ;Search result buffer (returned)
PUSH WORD ResultBufferLen ;Search result buffer length
CALL DosSearchPath
```

Returns WORD

### Example

The following example scans the environment segment for the PATH variable and prints its value. It then searches the path given by inserting the current directory into the value of the PATH variable for the file named "cmd.exe", and prints the full filename.

```
#define INCL_DOS

#include <os2.h>

#define ENVVARNAME "PATH" /* Environment variable name */
#define FILENAME "cmd.exe" /* File for which to search */
#define SEARCH_CUR_DIRECTORY 0x03 /* Search control - current dir., */
/* then environment variable */

main()
{
    PSZ FAR *ResultPointer; /* Environment scan result pointer (returned) */
    BYTE ResultBuffer[128]; /* Path search result (returned) */
    USHORT rc; /* return code */

    /** Scan environment segment for PATH; notice the far-string pointer **/
    /** specification ("%Fs") used to print. **/

    if(!rc=DosScanEnv(ENVVARNAME, /* Environment variable name */
                     &ResultPointer)) /* Scan result pointer (returned) */
        printf("%s is %Fs\n", ENVVARNAME, ResultPointer);

    /** Search current directory + PATH variable for "cmd.exe" **/
    if(!rc=DosSearchPath(SEARCH_CUR_DIRECTORY, /* Search control vector */
                        ENVVARNAME, /* Search path reference string */
                        FILENAME, /* File name string */
                        ResultBuffer, /* Search result (returned) */
                        sizeof(ResultBuffer))) /* Length of search result */
        printf("Found desired file -- %s\n", ResultBuffer);
}
```

---

This call selects the drive specified as the default drive for the calling process.

<b>DosSelectDisk (DriveNumber)</b>
------------------------------------

## Parameters

**DriveNumber** (*USHORT*) — input

New default drive number, where 1 = A and 2 = B and so on.

**rc** (*USHORT*) — return

Return code descriptions are:

0	NO_ERROR
15	ERROR_INVALID_DRIVE

## C Language

```
#define INCL_DOSFILEMGR
```

```
USHORT rc = DosSelectDisk(DriveNumber);
```

```
USHORT      DriveNumber; /* Default drive number */
```

```
USHORT      rc;          /* return code */
```

## Assembler Language

```
EXTRN DosSelectDisk:FAR
```

```
INCL_DOSFILEMGR EQU 1
```

```
PUSH WORD DriveNumber ;Default drive number
```

```
CALL DosSelectDisk
```

Returns WORD

# DosSelectSession – Select Foreground Session

---

This call allows a parent session to switch one of its child sessions to the foreground.

<b>DosSelectSession (SessID, Reserved)</b>
--

## Parameters

**SessID (USHORT)** – input

ID of the session to be switched to the foreground. The values and their meanings are:

Value	Definition
0	Switch the caller (the parent session) to the foreground.
≠0	Switch the child session to the foreground. The nonzero value specified for SessID must have been returned by a previous DosStartSession request.

**Reserved (ULONG)** – input

A DWord of 0.

**rc (USHORT)** – return

Return code descriptions are:

0	NO_ERROR
369	ERROR_SMG_INVALID_SESSION_ID
418	ERROR_SMG_INVALID_CALL
451	ERROR_SMG_SESSION_NOT_FOREGRND
452	ERROR_SMG_SESSION_NOT_PARENT
459	ERROR_SMG_BAD_RESERVE

## Remarks

DosSelectSession can be issued by a parent session to select only itself or a child session. It cannot be used to select a grandchild session. The child session must have been started by the caller with a request to DosStartSession, specifying "Related = 1." Sessions started as independent sessions cannot be selected with this call.

When DosSelectSession is issued, the session is brought to the foreground only if the parent session or one of its descendant sessions is executing in the foreground. Otherwise, ERROR\_SMG\_SESSION\_NOT\_FOREGRND is returned.

## C Language

```
#define INCL_DOSSESMGR

USHORT rc = DosSelectSession(SessID, Reserved);

USHORT SessID; /* Session ID */
ULONG 0; /* Reserved (must be zero) */

USHORT rc; /* return code */
```

## Assembler Language

```
EXTRN DosSelectSession:FAR
INCL_DOSSESMGR EQU 1

PUSH WORD SessID ;Session ID
PUSH DWORD 0 ;Reserved (must be zero)
CALL DosSelectSession

Returns WORD
```

# DosSemClear — Clear Semaphore

This call unconditionally clears a semaphore. If any threads were blocked on the semaphore, they are restarted.

**DosSemClear (SemHandle)**

## Parameters

**SemHandle (HSEM)** — input

Reference to the semaphore.

For a system semaphore, this reference is the handle returned by a DosCreateSem or DosOpenSem request that granted the requesting thread access to the semaphore.

For a RAM semaphore, this reference is the address of a doubleword of storage, allocated and initialized to zero by the application. This sets the semaphore as unowned. Other than initializing the doubleword to zero, an application must not modify a RAM semaphore directly; instead it manipulates the semaphore with semaphore function calls.

**rc (USHORT)** — return

Return code descriptions are:

0	NO_ERROR
101	ERROR_EXCL_SEM_ALREADY_OWNED

## Remarks

DosSemClear typically is used to release a semaphore obtained through DosSemRequest. DosSemClear also is used with the semaphore signaling functions DosSemSetWait, DosSemWait, and DosMuxSemWait.

If the semaphore is an exclusive system semaphore, it has a use count associated with it, which is incremented by a DosSemRequest and decremented by a DosSemClear. The semaphore is not actually cleared and made available to the next thread that wants to access the resource until the semaphore has been cleared the same number of times it has been requested. This means that exclusive system semaphores can be used in recursive routines. When the use count is 0, the semaphore is available to be claimed by the next user of the protected resource.

Normally, DosSemClear cannot be issued against a system semaphore owned by another process unless the NoExclusive option is set by the DosCreateSem request that created the semaphore. However, at interrupt time any thread may clear an exclusively owned semaphore.

## C Language

```
#define INCL_DOSSEMAPHORES

USHORT rc = DosSemClear(SemHandle);

HSEM      SemHandle;    /* Semaphore handle */

USHORT     rc;           /* return code */
```

## Assembler Language

```
EXTRN DosSemClear:FAR
INCL_DOSSEMAPHORES EQU 1

PUSH  DWORD SemHandle    ;Semaphore handle
CALL  DosSemClear

Returns WORD
```

# DosSemClear — Clear Semaphore

## Example

The following example illustrates the serialization of access to a shared resource between threads of the same process. The program creates a nonexclusive system semaphore named `resource.sem`, requests access to the semaphore, clears it, and finally closes the semaphore. For an illustration of notification of events, see the example given in `DosOpenSem`, `DosSemSet`, or `DosSemWait`.

```
#define INCL_DOSSEMAPHORES

#include <os2.h>

#define SEM_NAME "\\SEM\\resource.sem" /* Semaphore name */
#define TIMEOUT 1500L                 /* Timeout (in milliseconds) */

main()
{
    HSEM SemHandle;
    USHORT rc;

    /* Note: the semaphore could have been created by another */
    /* thread. */
    DosCreateSem(CSEM_PUBLIC, /* Ownership - nonexclusive */
                &SemHandle, /* Semaphore handle (returned) */
                SEM_NAME); /* Semaphore name */
    if(!(rc = DosSemRequest(SemHandle, /* Semaphore Handle */
                           TIMEOUT))) /* Timeout Period */
    {
        /* Semaphore obtained; resource may now be used. */
        /* Clear the semaphore after using resource. */
        if(DosSemClear(SemHandle))
        {
            /* Semaphore exclusively owned by another process -- */
            /* cannot clear now. */
        }
    }
    else
    {
        /* Semaphore not obtained: error processing (i.e. switch on rc) */
    }
    /* Semaphore no longer needed; close it */
    if(DosCloseSem(SemHandle))
    {
        /* Semaphore is still set -- cannot close now */
    }
}
```

# DosSemRequest – Request Semaphore

This call obtains a semaphore and sets the semaphore as owned by the thread that requested it.

**DosSemRequest (SemHandle, Timeout)**

## Parameters

**SemHandle (HSEM)** – input

Reference to the semaphore.

For a system semaphore, this reference is the handle returned by a DosCreateSem or DosOpenSem request that granted the requesting thread access to the semaphore.

For a RAM semaphore, this reference is the address of a doubleword of storage, allocated and initialized to zero by the application. This sets the semaphore as unowned. Other than initializing the doubleword to zero, an application must not modify a RAM semaphore directly; instead it manipulates the semaphore with semaphore function calls.

**Timeout (LONG)** – input

Action taken by the requesting thread when the semaphore is owned by another thread. The values that can be specified are:

Value	Definition
-1	The requesting thread waits indefinitely for the semaphore to be cleared.
0	The requesting thread returns immediately.
> 0	The requesting thread waits the indicated number of milliseconds for the semaphore to be cleared before resuming execution.

**rc (USHORT)** – return

Return code descriptions are:

0	NO_ERROR
95	ERROR_INTERRUPT
100	ERROR_TOO_MANY_SEMAPHORES
105	ERROR_SEM_OWNER_DIED
121	ERROR_SEM_TIMEOUT

## Remarks

Typically, DosSemRequest is called to set a semaphore for the purpose of accessing a protected resource. DosSemRequest checks the status of the semaphore. If the semaphore is not set (that is, not owned) by another thread, DosSemRequest sets the semaphore as owned by the current thread and returns immediately to the caller.

If the semaphore is already owned by another thread, DosSemRequest optionally can block the requesting thread until the semaphore becomes available. The unblocking of a thread blocked by a DosSemRequest is level-triggered. That is, DosSemRequest does not return until the semaphore remains clear long enough for the affected thread to be redispached and successfully claim the semaphore. Thus, if a number of threads are blocked indefinitely on DosSemRequest calls for the semaphore, only the thread that sets the semaphore is actually unblocked. If a milliseconds value is specified for the Timeout parameter, this places an upper limit on the amount of time the requesting thread remains blocked, waiting for the semaphore to become available.

When a thread no longer requires the protected resource, it issues DosSemClear to clear the semaphore, so another thread may claim it with a successful DosSemRequest. If the semaphore is an exclusive system semaphore, it has a use count associated with it, which is incremented by a DosSemRequest and decremented by a DosSemClear. The semaphore is not actually cleared and made available to the next thread that wants to access the resource until the semaphore has been cleared the same number of times it has been requested. This means that exclusive system semaphores can be used in recursive routines. When the use count is 0, the semaphore is available to be claimed by the next user of the protected resource.

# DosSemRequest — Request Semaphore

If a process terminates while owning a nonexclusive system semaphore, `ERROR_SEM_OWNER_DIED` is returned to the thread that next gets the semaphore through `DosSemRequest`. That thread takes steps to ensure the integrity of the resource. The thread can release the resource by issuing a `DosSemClear`, or it can reset the `ERROR_SEM_OWNER_DIED` return code condition flagged in the semaphore data structure.

**Note:** To request a Fast-Safe RAM semaphore, a thread calls `DosFSRamSemRequest`.

## C Language

```
#define INCL_DOSSEMAPHORES

USHORT rc = DosSemRequest(SemHandle, Timeout);

HSEM      SemHandle;    /* Semaphore handle */
LONG      Timeout;      /* Timeout (in milliseconds) */

USHORT     rc;           /* return code */
```

## Assembler Language

```
EXTRN DosSemRequest:FAR
INCL_DOSSEMAPHORES EQU 1

PUSH    DWORD    SemHandle    ;Semaphore handle
PUSH    DWORD    Timeout      ;Timeout (in milliseconds)
CALL    DosSemRequest

Returns WORD
```

## Example

The following example illustrates the serialization of access to a shared resource between threads of the same process. The program creates a nonexclusive system semaphore named `resource.sem`, requests access to the semaphore, clears it, and finally closes the semaphore. For an illustration of notification of events, see the example given in `DosOpenSem`, `DosSemSet`, or `DosSemWait`.

## DosSemRequest – Request Semaphore

```
#define INCL_DOSSEMAPHORES

#include <os2.h>

#define SEM_NAME "\\SEM\\resource.sem" /* Semaphore name */
#define TIMEOUT 1500L /* Timeout (in milliseconds) */

main()
{
    HSEM SemHandle;
    USHORT rc;

    /* Note: the semaphore could have been created by another */
    /* thread. */

    DosCreateSem(CSEM_PUBLIC, /* Ownership - nonexclusive */
                &SemHandle, /* Semaphore handle (returned) */
                SEM_NAME); /* Semaphore name */
    if(!rc = DosSemRequest(SemHandle, /* Semaphore Handle */
                          TIMEOUT)) /* Timeout Period */
    {
        /* Semaphore obtained; resource may now be used. */
        /* Clear the semaphore after using resource. */

        if(DosSemClear(SemHandle))
        {
            /* Semaphore exclusively owned by another process -- */
            /* cannot clear now. */
        }
    }
    else
    {
        /* Semaphore not obtained: error processing (i.e. switch on rc) */
    }
    /* Semaphore no longer needed; close it */
    if(DosCloseSem(SemHandle))
    {
        /* Semaphore is still set -- cannot close now */
    }
}
```



# DosSemSet — Set Semaphore Owned

This call unconditionally sets a semaphore; that is, it sets the semaphore whether or not it is already set.

<b>DosSemSet (SemHandle)</b>
------------------------------

## Parameters

**SemHandle (HSEM)** — input

Reference to the semaphore.

For a system semaphore, this reference is the handle returned by a DosCreateSem or DosOpenSem request that granted the requesting thread access to the semaphore.

For a RAM semaphore, this reference is the address of a doubleword of storage, allocated and initialized to zero by the application. This sets the semaphore as unowned. Other than initializing the doubleword to zero, an application must not modify a RAM semaphore directly; instead it manipulates the semaphore with semaphore function calls.

**rc (USHORT)** — return

Return code descriptions are:

0	NO_ERROR
100	ERROR_TOO_MANY_SEMAPHORES
103	ERROR_TOO_MANY_SEM_REQUESTS

## Remarks

DosSemSet usually is not required in a resource control environment using DosSemRequest and DosSemClear. It typically is used in a signaling environment implemented with DosSemWait, DosMuxSemWait, and DosSemClear. These function calls are used to block one or more threads on a set semaphore and awaken them when an event occurs.

DosSemSet cannot be issued against a system semaphore that is owned by another thread, unless the NoExclusive option was set in the original DosCreateSem request.

## C Language

```
#define INCL_DOSSEMAPHORES
```

```
USHORT rc = DosSemSet(SemHandle);
```

```
HSEM SemHandle; /* Semaphore handle */
```

```
USHORT rc; /* return code */
```

## Assembler Language

```
EXTRN DosSemSet:FAR
```

```
INCL_DOSSEMAPHORES EQU 1
```

```
PUSH DWORD SemHandle ;Semaphore handle
```

```
CALL DosSemSet
```

```
Returns WORD
```

## DosSemSet — Set Semaphore Owned

### Example

The following example illustrates the notification of an event between threads of different processes. Process1 creates a nonexclusive system semaphore named process1.sem and sets it. It then invokes process2 to run asynchronously. Process1 then waits, with a time out of 4.5 seconds, for process2 to open the semaphore and clear it, thereby notifying process1 to resume. Notice that there is a small possibility of process1's missing the notification because process2 may clear the semaphore before process1 issues DosSemWait. See the example for DosSemSetWait for an alternative that would correct this.

```
/* ----- process1.c ---- */

#define INCL_DOSSEMAPHORES
#define INCL_DOSPROCESS

#include <os2.h>

#define SEM_NAME "\\SEM\\process1.sem" /* Semaphore name */
#define TIMEOUT 4500L                /* Timeout period */
#define START_PROGRAM "process2.exe" /* Name of program file */

main()
{
    HSEM      SemHandle;
    CHAR      ObjFail [50];
    PSZ       Args;
    PSZ       Envs;
    RESULTCODES ReturnCodes;
    USHORT    rc;

    printf("Process1 now running. \n");
    rc = DosCreateSem(CSEM_PUBLIC, /* Ownership - nonexclusive */
                    &SemHandle, /* Semaphore handle (returned) */
                    SEM_NAME); /* Semaphore name string */
    printf("Process1.sem created; return code is %d \n", rc);

    /*** SET the semaphore. ***/
    if((rc = DosSemSet(SemHandle))) /* Semaphore handle */

    /***/
    {
        /* Cannot set semaphore -- error processing */
    }
    /* Start a separate process */
    if(! (DosExecPgm(ObjFail, /* Object name buffer */
                    sizeof(ObjFail), /* Length of obj. name buffer */
                    EXEC_ASYNC, /* Execution flag - asynchronous */
                    Args, /* Ptr. to argument string */
                    Envs, /* Ptr. to environment string */
                    &ReturnCodes, /* Ptr. to resultcodes struct. */
                    START_PROGRAM))) /* Name of program file */
        printf("Process2 started. Process1 will try to wait for notification. \n");

    /*** WAIT for a notification from process2 on process1.sem ***/
    if((rc = DosSemWait(SemHandle, /* Semaphore handle */
                       TIMEOUT))) /* Timeout period */

    /***/
    {
        /* No notification (either interrupt or timeout); error processing */
    }
    else
    {
        /* Notification received. Normal processing */
        printf("Process2 cleared semaphore; process1 running again. \n");
    }
}
```

## DosSemSet — Set Semaphore Owned

```
/* ----- process2.c -----*/

#define INCL_DOSSEMAPHORES

#include <os2.h>

#define SEM_NAME "\\SEM\\process1.sem"    /* Semaphore name */

main()
{
    HSEM SemHandle;
    USHORT rc;

    /* Obtain access to semaphore created by process1 via OPEN */
    if((rc=DosOpenSem(&SemHandle,          /* Semaphore handle (returned) */
                     SEM_NAME)))          /* Semaphore Name */
    {
        /* Could not open -- error processing (switch on rc). */
    }
    else
    {
        /* Opened semaphore; normal processing. Clear semaphore when done. */
        printf("Semaphore OPENED. \n");
        if(!(rc=DosSemClear(SemHandle)))    /* Semaphore handle */
            printf("Semaphore CLEARED. \n");
    }
}
```

# DosSemSetWait – Set Semaphore and Wait for Next Clear

This call sets a semaphore if the semaphore is not already set and waits until another thread clears the semaphore with a DosSemClear call.

**DosSemSetWait (SemHandle, Timeout)**

## Parameters

**SemHandle (HSEM)** – input  
Reference to the semaphore.

For a system semaphore, this reference is the handle returned by a DosCreateSem or DosOpenSem request that granted the requesting thread access to the semaphore.

For a RAM semaphore, this reference is the address of a doubleword of storage, allocated and initialized to zero by the application. This sets the semaphore as unowned. Other than initializing the doubleword to zero, an application must not modify a RAM semaphore directly; instead it manipulates the semaphore with semaphore function calls.

**Timeout (LONG)** – input  
Action taken by the requesting thread when the semaphore is set. The values that can be specified are:

Value	Definition
-1	The requesting thread waits indefinitely for the semaphore to be cleared.
0	The requesting thread returns immediately.
> 0	The requesting thread waits the indicated number of milliseconds for the semaphore to be cleared before resuming execution.

**rc (USHORT)** – return  
Return code descriptions are:

0	NO_ERROR
95	ERROR_INTERRUPT
101	ERROR_EXCL_SEM_ALREADY_OWNED
103	ERROR_TOO_MANY_SEM_REQUESTS
121	ERROR_SEM_TIMEOUT

## Remarks

DosSemSetWait combines the functions of DosSemSet and DosSemWait and is used when there is a chance the semaphore may be cleared by a thread that gets an intervening time slice between calls by the current thread to set the semaphore and wait until it is cleared.

A DosSemSetWait request differs from a DosSemWait request in that it ensures that the semaphore is set so that it can block on it. Issuing DosSemWait on a semaphore that has been cleared has no effect. Instead of blocking, the caller continues to execute.

The unblocking of a thread blocked by a DosSemSetWait is level-triggered. That is, DosSemSetWait does not return until the semaphore remains clear long enough for the affected thread to be redispached and determine that the semaphore is clear.

DosSemSetWait cannot be issued against a system semaphore owned by another thread unless the NoExclusive option was selected on the DosCreateSem request that created the semaphore.

# DosSemSetWait — Set Semaphore and Wait for Next Clear

## C Language

```
#define INCL_DOSSEMAPHORES

USHORT rc = DosSemSetWait(SemHandle, Timeout);

HSEM      SemHandle;    /* Semaphore handle */
LONG      Timeout;      /* Timeout (in milliseconds) */

USHORT     rc;          /* return code */
```

## Assembler Language

```
EXTRN DosSemSetWait:FAR
INCL_DOSSEMAPHORES EQU 1

PUSH  DWORD  SemHandle    ;Semaphore handle
PUSH  DWORD  Timeout      ;Timeout (in milliseconds)
CALL  DosSemSetWait

Returns WORD
```

## Example

The following example illustrates the notification of an event between threads of different processes. Process1 creates a nonexclusive system semaphore named process1.sem. It then invokes process2 to run asynchronously. Finally, process1 sets the semaphore and waits, with a timeout of 4.5 seconds, for process2 to open the semaphore and clear it. Process1 resumes after this clearance. The following example is different from the one given for DosSemSet and DosSemWait only in that the set and wait functions are performed in one atomic step, via DosSemSetWait. Hence, in the following example, there is no possibility for process1 to miss the notification because of a premature clear by process2.

## DosSemSetWait — Set Semaphore and Wait for Next Clear

```

/* ----- process1.c ---- */

#define INCL_DOSSEMAPHORES
#define INCL_DOSPROCESS

#include <os2.h>

#define SEM_NAME "\\SEM\\process1.sem" /* Semaphore name */
#define TIMEOUT 4500L                /* Timeout period */
#define START_PROGRAM "process2.exe" /* Name of program file */

main()
{
    HSEM      SemHandle;
    CHAR      ObjFail [50];
    PSZ       Args;
    PSZ       Envs;
    RESULTCODES ReturnCodes;
    USHORT    rc;

    printf("Process1 now running. \n");
    rc = DosCreateSem(CSEM_PUBLIC, /* Ownership indicator */
                    &SemHandle, /* Semaphore handle (returned) */
                    SEM_NAME); /* Semaphore name string */
    printf("Process1.sem created; return code is %d \n", rc);

    /* Start a separate process */
    if(!(DosExecPgm(ObjFail, /* Object name buffer */
                  sizeof(ObjFail), /* Length of obj. name buffer */
                  EXEC_ASYNC, /* Execution flag - asynchronous */
                  Args, /* Ptr. to argument string */
                  Envs, /* Ptr. to environment string */
                  &ReturnCodes, /* Ptr. to resultcodes struct. */
                  START_PROGRAM))) /* Name of program file */
        printf("Process2 started. Process1 will try to wait for notification. \n");

    /***** SET semaphore and WAIT for a notification *****/
    /***** from process2 on process1.sem *****/

    if((rc = DosSemSetWait(SemHandle, /* Semaphore handle */
                          TIMEOUT)) /* Timeout period */)

    /*****/
    {
        /* No notification (either interrupt or timeout); error processing */
    }
    else
    {
        /* Notification received. Normal processing */
        printf("Process2 cleared semaphore; process1 running again. \n");
    }
}

```

## DosSemSetWait — Set Semaphore and Wait for Next Clear

```
/* ----- process2.c -----*/

#define INCL_DOSSEMAPHORES

#include <os2.h>

#define SEM_NAME "\\SEM\\process1.sem"  /* Semaphore name */

main()
{
    HSEM SemHandle;
    USHORT rc;

    /* Obtain access to semaphore created by process1 via OPEN */
    if((rc=DosOpenSem(&SemHandle,      /* Semaphore handle (returned) */
                     SEM_NAME))      /* Semaphore Name */)
    {
        /* Could not open -- error processing (switch on rc). */
    }
    else
    {
        /* Opened semaphore; normal processing. Clear semaphore when done. */
        printf("Semaphore OPENED. \n");
        if(!(rc=DosSemClear(SemHandle))) /* Semaphore handle */
            printf("Semaphore CLEARED. \n");
    }
}
```

# DosSemWait – Wait for Semaphore To Clear

This call blocks the current thread until an indicated semaphore clears, but does not establish ownership of the semaphore.

**DosSemWait (SemHandle, Timeout)**

## Parameters

**SemHandle (HSEM)** – input

Reference to the semaphore.

For a system semaphore, this reference is the handle returned by a DosCreateSem or DosOpenSem request that granted the requesting thread access to the semaphore.

For a RAM semaphore, this reference is the address of a doubleword of storage, allocated and initialized to zero by the application. This sets the semaphore as unowned. Other than initializing the doubleword to zero, an application must not modify a RAM semaphore directly; instead it manipulates the semaphore with semaphore function calls.

**Timeout (LONG)** – input

Action taken by the requesting thread when the semaphore is set. The values that can be specified are:

Value	Definition
-1	The requesting thread waits indefinitely for the semaphore to be cleared.
0	The requesting thread returns immediately.
> 0	The requesting thread waits the indicated number of milliseconds for the semaphore to be cleared before resuming execution.

**rc (USHORT)** – return

Return code descriptions are:

0	NO_ERROR
95	ERROR_INTERRUPT
121	ERROR_SEM_TIMEOUT

## Remarks

The unblocking of a thread blocked by a DosSemWait is level-triggered. That is, DosSemWait does not return until the semaphore remains clear long enough for the affected thread to be redispached and determine that the semaphore is clear.

When an application needs to guarantee that another event has occurred before continuing, it calls DosSemSetWait. DosSemSetWait combines the functions of DosSemSet and DosSemWait and is used when there is a chance the semaphore may be cleared by a thread that gets an intervening time slice between calls by the current thread to set the semaphore and wait until it is cleared. Issuing DosSemWait on a semaphore that has been cleared has no effect; the thread continues to execute.

## C Language

```
#define INCL_DOSSEMAPHORES
```

```
USHORT rc = DosSemWait(SemHandle, Timeout);
```

```
HSEM      SemHandle;    /* Semaphore handle */
LONG      Timeout;      /* Timeout (in milliseconds) */

USHORT    rc;           /* return code */
```



# DosSemWait — Wait for Semaphore To Clear

## Assembler Language

```
EXTRN DosSemWait:FAR
INCL_DOSSEMAPHORES EQU 1

PUSH  DWORD  SemHandle    ;Semaphore handle
PUSH  DWORD  Timeout      ;Timeout (in milliseconds)
CALL  DosSemWait

Returns WORD
```

## Example

The following example illustrates the notification of an event between threads of different processes. Process1 creates a nonexclusive system semaphore named process1.sem and sets it. It then invokes process2 to run asynchronously. Process1 then waits, with a timeout of 4.5 seconds, for process2 to open the semaphore and clear it, thereby notifying process1 to resume. Notice that there is a small possibility of process1's missing the notification because process2 may clear the semaphore before process1 issues DosSemWait. See the example for DosSemSetWait for an alternative that would correct this.

## DosSemWait – Wait for Semaphore To Clear

```
/* ----- process1.c ----- */

#define INCL_DOSSEMAPHORES
#define INCL_DOSPROCESS

#include <os2.h>

#define SEM_NAME "\\SEM\\process1.sem" /* Semaphore name */
#define TIMEOUT 4500L /* Timeout period */
#define START_PROGRAM "process2.exe" /* Name of program file */

main()
{
    HSEM      SemHandle;
    CHAR      ObjFail [50];
    PSZ       Args;
    PSZ       Envs;
    RESULTCODES ReturnCodes;
    USHORT    rc;

    printf("Process1 now running. \n");
    rc = DosCreateSem(CSEM_PUBLIC, /* Ownership - nonexclusive */
                    &SemHandle, /* Semaphore handle (returned) */
                    SEM_NAME); /* Semaphore name string */
    printf("Process1.sem created; return code is %d \n", rc);

    /*** SET the semaphore. ***/
    if((rc = DosSemSet(SemHandle))) /* Semaphore handle */

    /*******
    {
        /* Cannot set semaphore -- error processing */
    }
    /* Start a separate process */
    if(!DosExecPgm(ObjFail, /* Object name buffer */
                  sizeof(ObjFail), /* Length of obj. name buffer */
                  EXEC_ASYNC, /* Execution flag - asynchronous */
                  Args, /* Ptr. to argument string */
                  Envs, /* Ptr. to environment string */
                  &ReturnCodes, /* Ptr. to resultcodes struct. */
                  START_PROGRAM)) /* Name of program file */
        printf("Process2 started. Process1 will try to wait for notification. \n");

    /*** WAIT for a notification from process2 on process1.sem ***/
    if((rc = DosSemWait(SemHandle, /* Semaphore handle */
                       TIMEOUT))) /* Timeout period */

    /*******
    {
        /* No notification (either interrupt or timeout); error processing */
    }
    else
    {
        /* Notification received. Normal processing */
        printf("Process2 cleared semaphore; process1 running again. \n");
    }
}
```

## DosSemWait — Wait for Semaphore To Clear

```
/* ----- process2.c -----*/

#define INCL_DOSSEMAPHORES

#include <os2.h>

#define SEM_NAME "\\SEM\\process1.sem"    /* Semaphore name */

main()
{
    HSEM SemHandle;
    USHORT rc;

    /* Obtain access to semaphore created by process1 via OPEN */
    if((rc=DosOpenSem(&SemHandle,          /* Semaphore handle (returned) */
                     SEM_NAME)))          /* Semaphore Name */

    /******
    {
        /* Could not open -- error processing (switch on rc). */
    }
    else
    {
        /* Opened semaphore; normal processing. Clear semaphore when done. */
        printf("Semaphore OPENED. \n");
        if(!rc=DosSemClear(SemHandle))    /* Semaphore handle */
            printf("Semaphore CLEARED. \n");
    }
}
```

# DosSendSignal – Send SIGINTR or SIGBREAK Signal

This call sends either a SIGINTR or a SIGBREAK signal to a process within a specified process tree.

**DosSendSignal (PID, SigNumber)**

## Parameters

**PID** (*USHORT*) – input

Root process ID of the subtree. It is not necessary that this process be alive, but it is necessary that this process be a direct child process of the process that issues this call.

**SigNumber** (*USHORT*) – input

Signal to send. It may be:

Value	Definition
1	(SIGINTR) Ctrl-C
4	(SIGBREAK) Ctrl-Break.

**rc** (*USHORT*) – return

Return code descriptions are:

0	NO_ERROR
1	ERROR_INVALID_FUNCTION
156	ERROR_SIGNAL_REFUSED
205	ERROR_NO_SIGNAL_SENT
209	ERROR_INVALID_SIGNAL_NUMBER
303	ERROR_INVALID_PROCID

## Remarks

The process that receives the signal is the most distant descendent process among those processes in the tree that have a handler installed for the signal.

The process tree is searched in descending order for processes having a handler installed for the specified signal. If there is at least one eligible process in the tree, the signal is sent to the process that is the most distant descendent process among the eligible processes. The selected process may have descendents, but none of them have the handler installed for the signal. If there is more than one most distant descendent eligible process, the signal is sent to one of them; which one is indeterminate.

Presentation Manager applications may not establish signal handlers for Ctrl-C and Ctrl-Break. Establishing a signal handler for Ctrl-C and Ctrl-Break is supported for VIO-Windowable and full-screen applications.

## C Language

```
#define INCL_DOSSIGNALS
```

```
USHORT rc = DosSendSignal(PID, SigNumber);
```

```
USHORT PID;          /* PID of root of subtree */  
USHORT SigNumber;    /* Signal Number to send */
```

```
USHORT rc;           /* return code */
```

# DosSendSignal — Send SIGINTR or SIGBREAK Signal

## Assembler Language

```
EXTRN DosSendSignal:FAR
INCL_DOSSIGNALS EQU 1

PUSH WORD PID ;PID of root of subtree
PUSH WORD SigNumber ;Signal Number to send
CALL DosSendSignal
```

Returns WORD

## Example

The following example illustrates the use of the Ctrl-C (SIGINTR) signal to signal time-critical events. Process1 invokes process2, which establishes a signal handler named CtrlC\_Handler() and waits, by blocking on a reserved RAM semaphore, for a signal from process1. A portion of process2 is immune to signalling.

```
#define INCL_DOSPROCESS
#define INCL_DOSSIGNALS

#include <os2.h>

#define SLEEPTIME 200L /* Sleep interval */
#define START_PROGRAM "process2.exe" /* Program name */

main()
{
    CHAR ObjFail[50];
    PSZ Args;
    PSZ Envs;
    RESULTCODES ReturnCodes;
    USHORT rc;

    /* Start process2 and check its PID */
    if(!DosExecPgm(ObjFail, /* Object name buffer */
                  sizeof(ObjFail), /* Length of obj. name buffer */
                  EXEC_ASYNC, /* Execution flag */
                  Args, /* Ptr. to argument string */
                  Envs, /* Ptr. to environment string */
                  &ReturnCodes, /* Ptr. to resultcodes struct. */
                  START_PROGRAM))) /* Name of program file */
        printf("Process2 started.\n");
    printf("Process2 ID is %d\n", ReturnCodes.codeTerminate);

    /* Sleep to give time slice to process2 */
    DosSleep(SLEEPTIME); /* Sleep interval */

    /*** After process2 sets signal handler, send process2 a signal ***/
    if(!rc = DosSendSignal(ReturnCodes.codeTerminate, /* PID of process2 */
                          SIG_CTRLC)) /* Signal to send */
        printf("Ctrl-C signal sent from Process1 to Process2.\n");
}
```

## DosSendSignal – Send SIGINTR or SIGBREAK Signal

```
/* ----- process2.c ----- */

#define INCL_DOSPROCESS
#define INCL_DOSSIGNALS
#define INCL_DOSERRORS

#include <os2.h>

#define SLEEPTIME      50L
#define TIMEOUT        5000L

VOID APIENTRY CtrlC_Handler(arg1, arg2)    /** Define signal handler **/
{
    USHORT      arg1;
    USHORT      arg2;
    {
        printf("Handler for Ctrl-C now running.\n");
        return;
    }
}

main()
{
    ULONG      RamSem = 0L;    /** Allocate and initialize Ram Semaphore */
    ULONG far  *RamSemHandle = &RamSem;    /** Ram Semaphore handle */
    USHORT      rc;

    /** Establish signal handler */
    if(!rc=DosSetSigHandler((PFNSIGHANDLER) CtrlC_Handler,
        NULL,                /** Previous handler - ignored */
        NULL,                /** Previous action - ignored */
        SIGA_ACCEPT,         /** Request type */
        SIG_CTRL_C))         /** Signal number */
        printf("Process2 has set Ctrl-C handler.\n");
    else
        /** Error processing on rc */;
    /** Get semaphore for first time */
    if(!rc=DosSemRequest(RamSemHandle,    /** Semaphore handle */
        TIMEOUT))                       /** Timeout interval */
        printf("Semaphore obtained.\n");

    /** Disable and then enable signal-handling */
    if(!rc=DosHoldSignal(HLDSIG_DISABLE)) /** Action code - disable */
    {
        printf("Signalling DISABLED.\n");

        /** Do signal-proof work here */
        if(!rc=DosHoldSignal(HLDSIG_ENABLE)) /** Action code - enable */
            printf("Signalling ENABLED.\n");
    }

    /** At this point, process1 may have sent a Ctrl-C signal. */
    /** Try to obtain semaphore again -- resulting in Timeout. */
    /** The Timeout, however, may be interrupted by the signal. */

    printf("Process2 will now wait on a Ramsem for a while.\n");
    if((rc=DosSemRequest(RamSemHandle,    /** Semaphore handle */
        TIMEOUT))                       /** Timeout interval */
        == ERROR_INTERRUPT)
        printf("Process2 interrupted while waiting, rc is %d.\n", rc);
}
```

# DosSetCp — Set Code Page

FAPI xPM

This call allows a process to set its code page and the session's display code page and keyboard code page.

<b>DosSetCp (CodePage, Reserved)</b>
--------------------------------------

## Parameters

**CodePage** (*USHORT*) — input

Code page identifier word that has one of the following values:

Value	Definition
437	IBM PC US 437 code page
850	Multilingual code page
860	Portuguese code page
863	Canadian-French code page
865	Nordic code page.

**Reserved** (*USHORT*) — input

Reserved must be set to zero.

**rc** (*USHORT*) — return

Return code descriptions are:

0	NO_ERROR
5	ERROR_ACCESS_DENIED
13	ERROR_INVALID_DATA
472	ERROR_INVALID_CODE_PAGE
482	ERROR_CP_SWITCH_INCOMPLETE

## Remarks

DosSetCp allows a program to set its code page. See CONFIG.SYS and the CODEPAGE command for preparing code pages for the system. The first code page specified in the CODEPAGE command is the default system code page. The session code page of a new session is set to the default system code page. A session's code page can be changed by the user with the CHCP command at the command prompt. The process code page of a new program started from a session command prompt is set to that session's code page.

DosSetCp sets the process code page of the calling process. The code page of a process is used in a series of ways. First, the printer code page is set to the process code page through the file system and printer spooler when the process makes an open printer request. Calling DosSetCp does not affect the code page of a printer opened prior to the call and does not affect the code page of a printer opened by another process. Second, country dependent information, by default, is retrieved encoded in the code page of the calling process. And third, a newly created process inherits its process code page from its parent process.

DosSetCp also sets, in the session to which the calling process belongs, the code page for the session's default logical keyboard and automatically flushes the keyboard buffer. It also sets the display code page for the session's logical display. This setting of the code page for the session's default logical keyboard and display overrides any previous setting by DosSetCp, DosSetProcCp, KbdSetCp, and VioSetCp by any process in the same session. Also see DosSetProcCp.

**C Language**

```
#define INCL_DOSNLS
```

```
USHORT rc = DosSetCp(CodePage, Reserved);
```

```
USHORT      CodePage;    /* Code page identifier */  
USHORT      0;           /* Reserved, set to zero */
```

```
USHORT      rc;           /* return code */
```

**Assembler Language**

```
EXTRN DosSetCp:FAR
```

```
INCL_DOSNLS      EQU 1
```

```
PUSH WORD CodePage    ;Code page identifier  
PUSH WORD 0           ;Reserved (must be zero)  
CALL DosSetCp
```

```
Returns WORD
```



# DosSetDateTime — Set Current Date and Time

FAPI

This call is used to set the date and time that are maintained by the operating system.

<b>DosSetDateTime</b> ( <i>DateTime</i> )
---

## Parameters

**DateTime** (*PDATETIME*) — input

Address of the date and time structure:

**hours** (*UCHAR*)

Current hour (0–23).

**minutes** (*UCHAR*)

Current minute (0–59).

**seconds** (*UCHAR*)

Current second (0–59).

**hundredths** (*UCHAR*)

Current hundredth of a second (0–99).

**day** (*UCHAR*)

Current day (1–31).

**month** (*UCHAR*)

Current month (1–12).

**year** (*USHORT*)

Current year (1980–2079).

**timezone** (*SHORT*)

Minutes west of UTC (Universal Time Coordinate –720 to 720).

**weekday** (*UCHAR*)

Current day of the week. This value is ignored and is calculated from the other parameter information.

**rc** (*USHORT*) — return

Return code descriptions are:

<b>0</b>	<b>NO_ERROR</b>
<b>327</b>	<b>ERROR_TS_DATETIME</b>

## Remarks

The value of **timezone** is the difference in minutes between the current time zone and UTC. This is a positive number if earlier than UTC, and negative number if it is later. For Eastern Standard Time this value is 300 (five hours earlier than UTC).

To get the date and time, issue **DosGetDateTime**. If the application is executing in the OS/2 environment, it is more efficient to obtain these variables by calling **DosGetInfoSeg** instead of this function. However, applications written to the family API cannot depend on the availability of **DosGetInfoSeg**.

Not adhering to the limits on any of the parameters results in the return code being set to **rc = 327** (**ERROR\_TS\_DATETIME**). Also, OS/2 verifies that the day is possible for the month and the year (even for leap year). If the day is not reasonable, OS/2 will also set **rc = 327**.

## C Language

```
typedef struct _DATETIME { /* date */
    UCHAR    hours;        /* current hour */
    UCHAR    minutes;      /* current minute */
    UCHAR    seconds;      /* current second */
    UCHAR    hundredths;   /* current hundredths of a second */
    UCHAR    day;          /* current day */
    UCHAR    month;        /* current month */
    USHORT   year;         /* current year */
    SHORT    timezone;     /* minutes of time west of UTC */
    UCHAR    weekday;      /* current day of week */
} DATETIME;

#define INCL_DOSDATETIME

USHORT rc = DosSetDateTime(DateTime);

PDATETIME    DateTime;    /* Date/time structure */
USHORT       rc;          /* return code */
```

## Assembler Language

```
DATETIME struc

    date_hours    db ? ;current hour
    date_minutes  db ? ;current minute
    date_seconds  db ? ;current second
    date_hundredths db ? ;current hundredths of a second
    date_day      db ? ;current day
    date_month    db ? ;current month
    date_year     dw ? ;current year
    date_timezone dw ? ;minutes of time west of UTC
    date_weekday  db ? ;current day of week

DATETIME ends

EXTRN DosSetDateTime:FAR
INCL_DOSDATETIME EQU 1

PUSH@ OTHER DateTime    ;Date/time structure
CALL DosSetDateTime
```

Returns WORD

## Example

The following example obtains and prints date and time information. It then changes the system date to 5/10/1987 and prints the updated information.

```
#define INCL_DOSDATETIME

#include <os2.h>

main()
{
    DATETIME    DateTime;    /* Structure to hold date/time info. */
    USHORT      rc;

    rc = DosGetDateTime(&DateTime);    /* Address of d/t structure */
    printf("Today is %d-%d-%d; the time is %d:%d\n", DateTime.month,
        DateTime.day, DateTime.year, DateTime.hours, DateTime.minutes);
    DateTime.day = 10;
    DateTime.month = 5;
    DateTime.year = 1987;
    printf("The new date is %d-%d-%d; the time is %d:%d\n", DateTime.month,
        DateTime.day, DateTime.year, DateTime.hours, DateTime.minutes);
    rc = DosSetDateTime(&DateTime);    /* Address of d/t structure */
    printf("rc is %d\n", rc);
}
```

This call sets the state of the specified file.

**DosSetFHandState** (**FileHandle**, **FileHandleState**)

## Parameters

**FileHandle** (*HFILE*) — input  
File handle to be set.

**FileHandleState** (*USHORT*) — input  
Contents of the open mode word defined in a previous DosOpen or DosOpen2 call.

Bit	Description
15	Zero Bit field. This bit must be set to zero.
14	Write-Through flag:  0 = Writes to the file may be run through the system buffer cache.  1 = Writes to the file may go through the system buffer cache, but the data is written (actual file I/O completed) before a synchronous write call returns. This state of the file defines it as a synchronous file. For synchronous files, this is a mandatory bit in that the data must be written out to the medium for synchronous write operations.  This bit is not inherited by child processes.
13	Fail-Errors flag. Media I/O errors are handled as follows:  0 = Reported through the system critical error handler.  1 = Reported directly to the caller by way of a return code.  Media I/O errors generated through an IOCTL category 8 function always get reported directly to the caller by way of a return code. The Fail-Errors function applies only to non-IOCTL handle-based type file I/O calls.  This bit is not inherited by child processes.
12	No-Cache/Cache flag. The file is opened as follows:  0 = It is advisable for the disk driver to cache the data in I/O operations on this file.  1 = I/O to the file need not be done through the disk driver cache.  This bit is an advisory bit, and is used to advise FSDs and device drivers on whether it is worth caching the data or not. This bit, like the write-through bit, is a per-handle bit.  This bit is not inherited by child processes.
11 – 8	Reserved Bits. These bits are reserved and should be set to the values returned by DosQFHandState in these positions.
7	Inheritance flag:  0 = File handle is inherited by a spawned process resulting from a DosExecPgm call.  1 = File handle is private to the current process.
6 – 4	Zero Bit field. These bits must be set to zero. Any other values are invalid.
3	Reserved Bit. This bit is reserved and should be set to the value returned by DosQFHandState for this position.
2 – 0	Zero Bit field. These bits must be set to zero. Any other values are invalid.

**rc** (*USHORT*) — return  
Return code descriptions are:

<b>0</b>	<b>NO_ERROR</b>
<b>6</b>	<b>ERROR_INVALID_HANDLE</b>
<b>87</b>	<b>ERROR_INVALID_PARAMETER</b>

**Remarks**

OS/2 does not guarantee the order that sectors are written for multiple sector writes. If an application requires several sectors written in a specific order, the operator should issue them as separate synchronous write operations. Setting the write-through flag does not affect any previous writes. That data may remain in the buffers. When a critical error occurs that the application cannot handle, it may reset critical error handling to be performed by the system. This is done by calling DosSetFHandState to turn off the fail/errors bit and reissuing the I/O call. The expected critical error reoccurs and control is passed to the system critical error handler. The precise time that the effect of this function is visible at the application level is unpredictable when asynchronous I/O is pending.

The file handle state bits set by this function can be queried by DosQFHandState.

**Family API Considerations**

Some options operate differently in the DOS mode than in OS/2 mode. Therefore, the following restrictions apply to DosSetFHandState when coding for the DOS mode:

- The validity of the file handle is not checked.
- The Inheritance flag must be set equal to zero. (This flag is not supported in DOS 2.X.)
- The Write-Through flag must be set equal to zero.
- The Fail-Errors flag must be set equal to zero.

**Named Pipe Considerations**

DosSetFHandState allows setting of the inheritance (I) and Write-Through (W) bits. Setting W to 1 prevents write-behind operations on remote pipes.

**C Language**

```
#define INCL_DOSFILEMGR

USHORT rc = DosSetFHandState(FileHandle, FileHandleState);

HFILE      FileHandle;      /* File handle */
USHORT      FileHandleState; /* File handle state */

USHORT      rc;              /* return code */
```

**Assembler Language**

```
EXTRN DosSetFHandState:FAR
INCL_DOSFILEMGR EQU 1

PUSH WORD FileHandle ;File handle
PUSH WORD FileHandleState ;File handle state
CALL DosSetFHandState

Returns WORD
```

---

This call sets attribute and extended attribute information for a file.

**DosSetFileInfo** (*FileHandle*, *FileInfoLevel*, *FileInfoBuf*, *FileInfoBufSize*)

### Parameters

**FileHandle** (*HFILE*) — input  
File handle.

**FileInfoLevel** (*USHORT*) — input  
Level of file information being set. A value of 1 or 2 can be specified. The structures described in *FileInfoBuf* indicate the information being set for each of these levels.

**FileInfoBuf** (*PBYTE*) — input  
Address of the storage area containing the structures for file information levels.

#### Level 1 Information

*FileInfoBuf* contains the following structure, to which information is specified in the following format:

##### filedate (*FDATE*)

Structure containing the date of file creation.

Bit	Description
15 – 9	Year, in binary, of file creation
8 – 5	Month, in binary, of file creation
4 – 0	Day, in binary, of file creation.

##### filetime (*FTIME*)

Structure containing the time of file creation.

Bit	Description
15 – 11	Hours, in binary, of file creation
10 – 5	Minutes, in binary, of file creation
4 – 0	Seconds, in binary number of two-second increments, of file creation.

##### fileaccessdate (*FDATE*)

Structure containing the date of last access. See *FDATE* in *filedate*.

##### fileaccesstime (*FTIME*)

Structure containing the time of last access. See *FTIME* in *filetime*.

##### writeaccessdate (*FDATE*)

Structure containing the date of last write. See *FDATE* in *filedate*.

##### writeaccesstime (*FTIME*)

Structure containing the time of last write. See *FTIME* in *filetime*.

##### filesize (*ULONG*)

File size.

##### filealloc (*ULONG*)

Allocated file size.

##### fileattrib (*USHORT*)

Attributes of the file, defined in *DosSetFileMode*.

#### Level 2 Information •

*FileInfoBuf* contains an EAOP structure, which has the following format:

##### lpGEAList (*PGEALIST*)

Address of GEAList. GEAList is a packed array of variable length “get EA” structures, each containing an EA name and the length of the name.

**fpFEAList (PFEALIST)**

Address of FEAList. FEAList is a packed array of variable length “full EA” structures, each containing an EA name and its corresponding value, as well as the lengths of the name and the value.

**oError (ULONG)**

Offset into structure where error has occurred.

Level 2 sets a series of EA name/value pairs. On input, FileInfoBuf is an EAOP structure above. fpGEAList is ignored. fpFEAList points to a data area where the relevant FEA list is to be found. oError is ignored. Following is the format of the FEAList structure:

**cbList (ULONG)**

Length of the FEA list, including the length itself.

**list (FEA)**

List of FEA structures. An FEA structure has the following format:

**Flags (BYTE)**

Bit indicator describing the characteristics of the EA being defined.

Bit	Description
15	Critical EA.
14 – 0	Reserved and must be set to zero.

If bit 15 is set to 1, this indicates a critical EA. If bit 15 is 0, this means the EA is noncritical; that is, the EA is not essential to the intended use by an application of the file with which it is associated.

**cbName (BYTE)**

Length of EA ASCII name, which does not include the null character.

**cbValue (USHORT)**

Length of EA value, which cannot exceed 64KB.

**szName (PSZ)**

Address of the ASCII name of EA.

**aValue (PSZ)**

Address of the free-format value of EA.

**Note:** The szName and aValue fields are not included as part of header or include files. Because of their variable lengths, these entries must be built manually.

On output, fpGEAList is unchanged. fpFEAList is unchanged as is the area pointed to by fpFEAList. If an error occurred during the set, oError is the offset of the FEA where the error occurred. The API return code is the error code corresponding to the condition generating the error. If no error occurred, oError is undefined.

**FileInfoBufSize (USHORT) – input**

Length of FileInfoBuf.

**rc (USHORT) – return**

Return code descriptions are:

0	NO_ERROR
1	ERROR_INVALID_FUNCTION
5	ERROR_ACCESS_DENIED
6	ERROR_INVALID_HANDLE
87	ERROR_INVALID_PARAMETER
122	ERROR_INSUFFICIENT_BUFFER
124	ERROR_INVALID_LEVEL
130	ERROR_DIRECT_ACCESS_HANDLE
254	ERROR_INVALID_EA_NAME
255	ERROR_EA_LIST_INCONSISTENT

### Remarks

DosSetFileInfo is successful only when the file is opened for write access, with a deny-both sharing mode specified for access to the file by other processes. If the file is already opened with conflicting sharing rights, the call to DosOpen or DosOpen2 will fail.

A 0 value in the date and time components of a field does not change the field. For example, if both "last write date" and "last write time" are specified as 0 in the Level 1 information structure, then both attributes of the file are left unchanged. If either "last write date" or "last write time" are specified as non-zero, both attributes of the file are set to the new values.

The FAT file system supports modification of only the dates and times of the last write. Creation and last access dates and times are not affected.

The last modification date and time will get changed if the extended attributes are modified.

### Family API Considerations

It is not possible to create a label with leading blank characters in DOS mode, because of restrictions on the previous Interrupt 21h function call (create an FCB type file), which must be used by FAPI.

### C Language

```
typedef struct _FDATE {    /* fdate */

    unsigned day   : 5;    /* binary day for directory entry */
    unsigned month : 4;    /* binary month for directory entry */
    unsigned year  : 7;    /* binary year for directory entry */

} FDATE;

typedef struct _FTIME {    /* ftime */

    unsigned twosecs : 5;    /* binary number of two-second increments */
    unsigned minutes : 6;    /* binary number of minutes */
    unsigned hours   : 5;    /* binary number of hours */

} FTIME;

typedef struct _FILESTATUS {    /* fsts */

    FDATE fdateCreation;    /* date of file creation */
    FTIME ftimeCreation;    /* time of file creation */
    FDATE fdateLastAccess;  /* date of last access */
    FTIME ftimeLastAccess;  /* time of last access */
    FDATE fdateLastWrite;   /* date of last write */
    FTIME ftimeLastWrite;   /* time of last write */
    ULONG cbFile;           /* file size (end of data) */
    ULONG cbFileAlloc;      /* file allocated size */
    USHORT attrFile;        /* attributes of the file */

} FILESTATUS;
```

```

typedef struct _GEA {      /* gea */

    BYTE cbName;          /* name length not including NULL */
    CHAR szName[1];       /* attribute name */

} GEA;

typedef struct _GEALIST {  /* geal */

    ULONG cbList;         /* total bytes of structure including full list */
    GEA list[1];          /* variable length GEA structures */

} GEALIST;

typedef struct _FEA {      /* fea */

    BYTE fEA;             /* flags */
    BYTE cbName;          /* name length not including NULL */
    USHORT cbValue;       /* value length */

} FEA;

typedef struct _FEALIST {  /* feal */

    ULONG cbList;         /* total bytes of structure including full list */
    FEA list[1];          /* variable length FEA structures */

} FEALIST;

typedef struct _EAOP {     /* eaop */

    PGEALIST fpGEAList;    /* general EA list */
    PFEALIST fpFEAList;    /* full EA list */
    ULONG oError;

} EAOP;

#define INCL_DOSFILEMGR

USHORT rc = DosSetFileInfo(FileHandle, FileInfoLevel, FileInfoBuf,
                           FileInfoBufSize);

HFILE      FileHandle;    /* File handle */
USHORT     FileInfoLevel; /* File info data required */
PBYTE      FileInfoBuf;   /* File info buffer */
USHORT     FileInfoBufSize; /* File info buffer size */

USHORT     rc;            /* return code */

```

**Assembler Language**



# DosSetFileInfo —

## Set File Information

FAPI

```
FDATE    struc
    fdate_fs dw ?
FDATE    ends

FTIME    struc
    ftime_fs dw ?
FTIME    ends

FILESTATUS struc
    fsts_fdateCreation dw (size FDATE)/2 dup (?) ;date of file creation
    fsts_ftimeCreation dw (size FTIME)/2 dup (?) ;time of file creation
    fsts_fdateLastAccess dw (size FDATE)/2 dup (?) ;date of last access
    fsts_ftimeLastAccess dw (size FTIME)/2 dup (?) ;time of last access
    fsts_fdateLastWrite dw (size FDATE)/2 dup (?) ;date of last write
    fsts_ftimeLastWrite dw (size FTIME)/2 dup (?) ;time of last write
    fsts_cbFile dd ? ;file size (end of data)
    fsts_cbFileAlloc dd ? ;file allocated size
    fsts_attrFile dw ? ;attributes of the file
FILESTATUS ends

GEA    struc
    gea_cbName db ? ;name length not including NULL
    gea_szName db 1 dup (?) ;attribute name
GEA    ends

GEALIST    struc
    geal_cbList dd ? ;total bytes of structure including full list
    geal_list db size GEA * 1 dup (?) ;variable length GEA structures
GEALIST    ends

FEA    struc
    fea_fEA db ? ;flags
    fea_cbName db ? ;name length not including NULL
    fea_cbValue dw ? ;value length
FEA    ends

FEALIST    struc
    feal_cbList dd ? ;total bytes of structure including full list
    feal_list db size FEA * 1 dup (?) ;variable length FEA structures
FEALIST    ends

EAOP    struc
    eaop_fpGEAList dd ? ;general EA list
    eaop_fpFEAList dd ? ;full EA list
    eaop_oError dd ? ;
EAOP    ends

EXTRN DosSetFileInfo:FAR
INCL_DOSFILEMGR EQU 1

PUSH WORD FileHandle ;File handle
PUSH WORD FileInfoLevel ;File info data required
PUSH@ OTHER FileInfoBuf ;File info buffer
PUSH WORD FileInfoBufSize ;File info buffer size
CALL DosSetFileInfo

Returns WORD
```

---

This call changes the mode (attribute) of the specified file.

**DosSetFileMode (FileName, NewAttribute, Reserved)**

## Parameters

**FileName (PSZ)** – input

Address of the file path name.

DosQSysInfo is called by an application during initialization to determine the maximum path length allowed by OS/2.

**NewAttribute (USHORT)** – input

File's new attribute. File attributes are defined as follows:

Bit	Description
15 – 6	Reserved and must be zero.
5	File archive
4	Subdirectory
3	Volume label
2	System file (excluded from normal directory searches)
1	Hidden file
0	Read only file

These bits may be set individually or in combination. For example, an attribute value of 0021H (bits 5 and 0 set to 1) indicates a read-only file that should be archived.

**Reserved (ULONG)** – input

Reserved must be set to zero.

**rc (USHORT)** – return

Return code descriptions are:

0	NO_ERROR
2	ERROR_FILE_NOT_FOUND
3	ERROR_PATH_NOT_FOUND
5	ERROR_ACCESS_DENIED
26	ERROR_NOT_DOS_DISK
32	ERROR_SHARING_VIOLATION
36	ERROR_SHARING_BUFFER_EXCEEDED
87	ERROR_INVALID_PARAMETER
108	ERROR_DRIVE_LOCKED
206	ERROR_FILENAME_EXCED_RANGE

## Remarks

Attributes for Volume Label (0008H) and Subdirectory (0010H) cannot be changed using DosSetFileMode. If these attributes are specified, ERROR\_INVALID\_PARAMETER is returned.

DosQFileMode is used to query the current settings for file attributes. Calling DosQFSInfo obtains volume label information.

Attributes of root directories cannot be changed using DosSetFileMode. If these attributes are specified, ERROR\_ACCESS\_DENIED is returned.

# DosSetFileMode —

## Set File Mode

FAPI

### C Language

```
#define INCL_DOSFILEMGR

USHORT rc = DosSetFileMode(FileName, NewAttribute, Reserved);

PSZ      FileName;      /* File path name string */
USHORT   NewAttribute;   /* New attribute of file */
ULONG    0;             /* Reserved (must be zero) */

USHORT    rc;            /* return code */
```

### Assembler Language

```
EXTRN DosSetFileMode:FAR
INCL_DOSFILEMGR EQU 1

PUSH@ ASCIIZ FileName      ;File path name string
PUSH WORD NewAttribute     ;New attribute of file
PUSH DWORD 0               ;Reserved (must be zero)
CALL DosSetFileMode

Returns WORD
```

This call sets information for a file system device.

**DosSetFSInfo (DriveNumber, FSInfoLevel, FSInfoBuf, FSInfoBufSize)**

## Parameters

**DriveNumber** (*USHORT*) — input

Logical drive number, for example, 0 = default and 1 = A, and the FSD for the media currently in that drive. A value of '0xFFFF' notes that FSInfoBuf contains the ASCIIZ path name of the FSD.

**FSInfoLevel** (*USHORT*) — input

Level of file information to be set. A value of 2 is the only valid value for FSInfoLevel.

**FSInfoBuf** (*PBYTE*) — input

Address of the storage area where the system gets the new file system information.

### Level 2 Information

Level 2 information is specified in the following format:

Byte	Description
1	Length of Volume Label (null not included)
2 – N	Volume Label ASCIIZ string.

**FSInfoBufSize** (*USHORT*) — input

Length of FSInfoBuf.

**rc** (*USHORT*) — return

Return code descriptions are:

0	NO_ERROR
15	ERROR_INVALID_DRIVE
82	ERROR_CANNOT_MAKE
122	ERROR_INSUFFICIENT_BUFFER
123	ERROR_INVALID_NAME
124	ERROR_INVALID_LEVEL
154	ERROR_LABEL_TOO_LONG

## Remarks

Trailing blanks supplied at volume label definition time are not returned by DosQFSInfo.

File system information can be set only if the volume is opened in a mode that allows write-access.

## C Language

```
#define INCL_DOSFILEMGR
```

```
USHORT rc = DosSetFSInfo(DriveNumber, FSInfoLevel, FSInfoBuf, FSInfoBufSize);
```

```
USHORT DriveNumber; /* Drive number */
USHORT FSInfoLevel; /* File system data type */
PBYTE FSInfoBuf; /* File system info buffer */
USHORT FSInfoBufSize; /* File system info buffer size */

USHORT rc; /* return code */
```

# DosSetFSInfo — Set File System Information

FAPI

## Assembler Language

```
EXTRN DosSetFSInfo:FAR
INCL_DOSFILEMGR      EQU 1

PUSH  WORD    DriveNumber    ;Drive number
PUSH  WORD    FSInfoLevel    ;File system data type
PUSH@ OTHER    FSInfoBuf      ;File system info buffer
PUSH  WORD    FSInfoBufSize  ;File system info buffer size
CALL  DosSetFSInfo
```

Returns WORD

## DosSetMaxFH – Set Maximum File Handles

---

This call defines the maximum number of file handles for the current process.

<b>DosSetMaxFH (NumberHandles)</b>
------------------------------------

### Parameters

**NumberHandles** (*USHORT*) – input

Total number of file handles to be provided.

**rc** (*USHORT*) – return

Return code descriptions are:

0	NO_ERROR
8	ERROR_NOT_ENOUGH_MEMORY
87	ERROR_INVALID_PARAMETER

### Remarks

OS/2 initially allots 20 file handles to a process, which is the recommended amount for an application. However, if the system limit has not been reached, this amount can be increased with DosSetMaxFH. When this call is made, all open file handles are preserved.

### C Language

```
#define INCL_DOSFILEMGR

USHORT rc = DosSetMaxFH(NumberHandles);

USHORT      NumberHandles; /* Number of file handles */

USHORT      rc;             /* return code */
```

### Assembler Language

```
EXTRN DosSetMaxFH:FAR
INCL_DOSFILEMGR EQU 1

PUSH WORD NumberHandles ;Number of file handles
CALL DosSetMaxFH

Returns WORD
```

# DosSetNmPHandState – Set Named Pipe Handle State

This call sets the read and blocking modes of a named pipe.

DosSetNmPHandState (Handle, PipeHandleState)

## Parameters

**Handle (HPIPE)** – input  
Handle of the named pipe returned by DosMakeNmPipe or DosOpen.

**PipeHandleState (USHORT)** – input  
Named pipe handle state, consists of the following bit fields:

Bit	Description
15	Blocking flag:  0 = Reads /Writes block if no data available.  1 = Reads/Writes return immediately if no data available.  Reads normally block until at least partial data can be returned. Writes by default block until all bytes requested have been written. Non-blocking mode (B = 1) changes this behavior as follows:  <b>DosRead</b> Returns BytesRead = 0 if no data is available. <b>DosWrite</b> Returns BytesWritten = 0 if there is not enough buffer space available in the pipe. Otherwise, the entire data area is transferred.
14 – 10	Reserved and must be set to zero.
9 – 8	Read Mode flag:  00 = Read pipe as byte stream.  01 = Read pipe as a message stream.
7 – 0	Reserved and must be set to zero.

**rc (USHORT)** – return  
Return code descriptions are:

0	NO_ERROR
87	ERROR_INVALID_PARAMETER
230	ERROR_BAD_PIPE
231	ERROR_PIPE_BUSY
233	ERROR_PIPE_NOT_CONNECTED

## Remarks

The read mode and blocking/non-blocking mode of a named pipe can be changed. The pipe mode can not be changed to non-blocking when another thread is blocked on an I/O to the same end of the pipe.

## C Language

```
#define INCL_DOSNMPIPES

USHORT rc = DosSetNmPHandState(Handle, PipeHandleState);

HPIPE      Handle;          /* Pipe handle */
USHORT     PipeHandleState; /* Pipe handle state */

USHORT     rc;              /* return code */
```

## **DosSetNmPHandState — Set Named Pipe Handle State**

### **Assembler Language**

```
EXTRN DosSetNmPHandState:FAR  
INCL_DOSNMPPIPES EQU 1
```

```
PUSH WORD Handle ;Pipe handle  
PUSH WORD PipeHandleState ;Pipe handle state  
CALL DosSetNmPHandState
```

Returns WORD



# DosSetNmPipeSem — Set Named Pipe Semaphore

---

This call attaches a system semaphore to a local named pipe.

**DosSetNmPipeSem (Handle, SemHandle, KeyHandle)**

## Parameters

**Handle (HPIPE)** — input

Handle of the named pipe returned by DosMakeNmPipe or DosOpen.

**SemHandle (HSEM)** — input

A system semaphore handle that is cleared when the pipe (identified by Handle) has read data or write space available.

**KeyHandle (USHORT)** — input

An arbitrary value that is used to associate named pipe actions with result codes in DosQNmPipeSemState.

**rc (USHORT)** — return

Return code descriptions are:

0	NO_ERROR
6	ERROR_INVALID_HANDLE
87	ERROR_INVALID_PARAMETER
120	ERROR_INVALID_FUNCTION
187	ERROR_SEM_NOT_FOUND
230	ERROR_BAD_PIPE
233	ERROR_PIPE_NOT_CONNECTED

## Remarks

This call is intended for use only with local named pipes. If an attempt is made to attach a semaphore to a remote named pipe, ERROR\_INVALID\_FUNCTION is returned.

DosSetNmPipeSem supports serving applications that need to handle a large number of incoming pipes. By using semaphores, the application avoids such resource-consuming methods as dedicating a thread for each incoming pipe, or polling each pipe with non-blocking I/O. Instead, the application issues a DosSemWait or a DosMuxSemWait and waits for notification that I/O can be performed.

The system semaphore indicated by SemHandle is attached to the named pipe indicated by Handle. Up to two semaphores may be attached to a pipe, one for the serving side and one for the client side. If a semaphore is already attached to a side of the pipe, the semaphore is overridden.

The arbitrary value assigned to KeyHandle as input for DosSetNmPipeSem is used by DosQNmPipeSemState as output. This enables the application to distinguish events specific to a particular pipe when several pipes are attached to the same semaphore. DosQNmPipeSemState can be used to provide additional information about the I/O that can be performed on the set of pipes.

## C Language

```
#define INCL_DOSNMPIPES
```

```
USHORT rc = DosSetNmPipeSem(Handle, SemHandle, KeyHandle);
```

```
HPIPE      Handle;      /* Pipe handle */
HSEM        SemHandle;   /* Semaphore handle */
USHORT      KeyHandle;    /* Key value */

USHORT      rc;           /* return code */
```

## **DosSetNmPipeSem — Set Named Pipe Semaphore**

### **Assembler Language**

```
EXTRN DosSetNmPipeSem:FAR
INCL_DOSNMPICES EQU 1

PUSH WORD Handle ;Pipe handle
PUSH DWORD SemHandle ;Semaphore handle
PUSH WORD KeyHandle ;Key value
CALL DosSetNmPipeSem

Returns WORD
```

# DosSetPathInfo – Set Information for a File or Directory

FAPI

This call sets attribute and extended attribute information for a file or subdirectory.

**DosSetPathInfo** (*PathName*, *PathInfoLevel*, *PathInfoBuf*, *PathInfoBufSize*, *PathInfoFlags*, *Reserved*)

## Parameters

**PathName** (*PSZ*) – input

Address of the ASCIIZ full path name of the file or subdirectory. Global file name characters are not permitted.

DosQSysInfo is called by an application during initialization to determine the maximum path length allowed by OS/2.

**PathInfoLevel** (*USHORT*) – input

Level of file object information being defined. A value of 1 or 2 can be specified. The structures described in PathInfoBuf indicate the information being set for each of these levels.

**PathInfoBuf** (*PBYTE*) – input

Address of the storage area containing the file information being set.

### Level 1 Information

PathInfoBuf contains the following structure, to which information is specified in the following format:

**filedate** (*FDATE*)

Structure containing the date of creation.

Bit	Description
15 – 9	Year, in binary, of creation
8 – 5	Month, in binary, of creation
4 – 0	Day, in binary, of creation.

**filetime** (*FTIME*)

Structure containing the time of creation.

Bit	Description
15 – 11	Hours, in binary, of creation
10 – 5	Minutes, in binary, of creation
4 – 0	Seconds, in binary number of two-second increments, of creation.

**fileaccessdate** (*FDATE*)

Structure containing the date of last access. See *FDATE* in filedate.

**fileaccesstime** (*FTIME*)

Structure containing the time of last access. See *FTIME* in filetime.

**writeaccessdate** (*FDATE*)

Structure containing the date of last write. See *FDATE* in filedate.

**writeaccesstime** (*FTIME*)

Structure containing the time of last write. See *FTIME* in filetime.

### Level 2 Information

PathInfoBuf contains an EAOP structure, which has the following format:

**fpGEAList** (*PGEALIST*)

Address of GEAList. GEAList is a packed array of variable length “get EA” structures, each containing an EA name and the length of the name.

**fpFEAList** (*PFEALIST*)

Address of FEAList. FEAList is a packed array of variable length “full EA” structures, each containing an EA name and its corresponding value, as well as the lengths of the name and the value.

## DosSetPathInfo — Set Information for a File or Directory

**oError (ULONG)**

Offset into structure where error has occurred.

Level 2 sets a series of EA name/value pairs. On input, FileInfoBuf is an EAOP structure above. fpGEAList is ignored. fpFEAList points to a data area where the relevant FEA list is to be found. oError is ignored. Following is the format of the FEAList structure:

**cbList (ULONG)**

Length of the FEA list, including the length itself.

**lList (FEA)**

List of FEA structures. An FEA structure has the following format:

**Flags (BYTE)**

Bit indicator describing the characteristics of the EA being defined.

Bit	Description
-----	-------------

15	Critical EA.
----	--------------

14 – 0	Reserved and must be set to zero.
--------	-----------------------------------

If bit 15 is set to 1, this indicates a critical EA. If bit 15 is 0, this means the EA is noncritical; that is, the EA is not essential to the intended use by an application of the file with which it is associated.

**cbName (BYTE)**

Length of EA ASCIIZ name, which does not include the null character.

**cbValue (USHORT)**

Length of EA value, which cannot exceed 64KB.

**szName (PSZ)**

Address of the ASCIIZ name of EA.

**aValue (PSZ)**

Address of the free-format value of EA.

**Note:** The szName and aValue fields are not included as part of header or include files. Because of their variable lengths, these entries must be built manually.

On output, fpGEAList is unchanged. fpFEAList is unchanged as is the area pointed to by fpFEAList. If an error occurred during the set, oError is the offset of the FEA where the error occurred. The API return code is the error code corresponding to the condition generating the error. If no error occurred, oError is undefined.

**PathInfoBufSize (USHORT) — input**

Length of PathInfoBuf.

**PathInfoFlags (USHORT) — input**

PathInfoFlags contain information on how the set operation is performed. Only one bit is defined. If PathInfoFlags = 0010H, then the information being set must be written out to disk before returning to the application. This guarantees that the EAs have been written to disk. All other bits are reserved and must be zero.

**Reserved (ULONG) — input**

Reserved, must be set to zero.

**rc (USHORT) — return**

Return code descriptions are:

0	NO_ERROR
32	ERROR_SHARING_VIOLATION
87	ERROR_INVALID_PARAMETER
124	ERROR_INVALID_LEVEL
206	ERROR_FILENAME_EXCED_RANGE
122	ERROR_INSUFFICIENT_BUFFER
254	ERROR_INVALID_EA_NAME
255	ERROR_EA_LIST_INCONSISTENT

# DosSetPathInfo —

## Set Information for a File or Directory

FAPI

### Remarks

To set any level of file information for a file or subdirectory with DosSetPathInfo, a process must have exclusive write access to the closed file object. Thus, if the file object is already accessed by another process, a call to DosSetPathInfo fails.

A zero (0) value in both the date and time components of a field cause that field to be left unchanged. For example, if both "Last write date" and "Last write time" are specified as zero in the Level 0001H information structure, then both attributes of the file are left unchanged. If either "Last write date" or "Last write time" are specified as non-zero, then both attributes of the file are set to the new values.

The write-through bit in PathInfo Flags should be used only in cases where it is necessary, for data integrity purposes, to write the EAs out to disk immediately, instead of caching them and writing them out later. Setting the write-through bit all the time may degrade the performance.

The last modification date and time will get changed if the extended attributes are modified.

### C Language

```
typedef struct _GEA {          /* gea */

    BYTE cbName;              /* name length not including NULL */
    CHAR szName[1];           /* attribute name */

} GEA;

typedef struct _GEALIST {      /* geal */

    ULONG cbList;             /* total bytes of structure including full list */
    GEA list[1];              /* variable length GEA structures */

} GEALIST;

typedef struct _FEA {          /* fea */

    BYTE fEA;                 /* flags */
    BYTE cbName;              /* name length not including NULL */
    USHORT cbValue;           /* value length */

} FEA;

typedef struct _FEALIST {      /* feal */

    ULONG cbList;             /* total bytes of structure including full list */
    FEA list[1];              /* variable length FEA structures */

} FEALIST;

typedef struct _EAOP {         /* eaop */

    PGEALIST fpGEAList;        /* general EA list */
    PFEALIST fpFEAList;        /* full EA list */
    ULONG oError;

} EAOP;

#define INCL_DOSFILEMGR

USHORT rc = DosSetPathInfo(PathName, PathInfoLevel, PathInfoBuf, PathInfoBufSize, PathInfoFlags, 0);

PSZ      PathName;           /* File or directory path name string */
USHORT   PathInfoLevel;      /* Info data type */
PBYTE    PathInfoBuf;        /* Info buffer */
USHORT   PathInfoBufSize;    /* Info buffer size */
USHORT   PathInfoFlags;      /* Path info flags */
ULONG    0;                  /* Reserved (must be zero) */

USHORT   rc;                 /* return code */
```

# DosSetPathInfo — Set Information for a File or Directory

## Assembler Language

```

GEA    struc

    gea_cbName    db  ?           ;name length not including NULL
    gea_szName    db  1 dup (?)   ;attribute name

GEA    ends

GEALIST    struc

    geal_cbList    dd  ?           ;total bytes of structure including full list
    geal_list      db  size GEA * 1 dup (?) ;variable length GEA structures

GEALIST    ends

FEA    struc

    fea_fEA        db  ? ;flags
    fea_cbName     db  ? ;name length not including NULL
    fea_cbValue    dw  ? ;value length

FEA    ends

FEALIST    struc

    feal_cbList    dd  ?           ;total bytes of structure including full list
    feal_list      db  size FEA * 1 dup (?) ;variable length FEA structures

FEALIST    ends

EAOP    struc

    eaop_fpGEAList dd  ? ;general EA list
    eaop_fpFEAList dd  ? ;full EA list
    eaop_oError    dd  ? ;

EAOP    ends

EXTRN DosSetPathInfo:FAR
INCL_DOSFILEMGR    EQU 1

PUSH@    ASCIIZ PathName        ;File or directory path name string
PUSH     WORD    PathInfoLevel   ;Info data type
PUSH@    OTHER    PathInfoBuf     ;Info buffer
PUSH     WORD    PathInfoBufSize ;Info buffer size
PUSH     WORD    Flags           ;Path info flags
PUSH     DWORD    0              ;Reserved (must be zero)
CALL     DosSetPathInfo

Returns WORD

```

# DosSetProcCp – Set Process Code Page

This call allows a process to set its code page.

<b>DosSetProcCp (CodePage, Reserved)</b>
--

## Parameters

**CodePage (USHORT)** – input

Code page identifier word that has one of the following values:

Value	Definition
437	IBM PC US 437 code page
850	Multilingual code page
860	Portuguese code page
863	Canadian-French code page
865	Nordic code page.

**Reserved (USHORT)** – input

Reserved must be set to zero.

**rc (USHORT)** – return

Return code description is:

0	NO_ERROR
472	ERROR_INVALID_CODEPAGE

## Remarks

DosSetProcCp sets the process code page of the calling process. The code page of a process is used in the following ways. First, the printer code page is set to the process code page through the file system and printer spooler when the process makes an open printer request. Calling DosSetProcCp does not affect the code page of a printer opened prior to the call and does not affect the code page of a printer opened by another process. Second, country dependent information, by default, is retrieved encoded in the code page of the calling process. And third, a newly created process inherits its process code page from its parent process. DosSetProcCp does not affect the display or keyboard code page. Also see DosSetCp.

## C Language

```
#define INCL_DOSNLS
```

```
USHORT rc = DosSetProcCp(CodePage, Reserved);
```

```
USHORT      CodePage;    /* Code page identifier */
USHORT      0;           /* Reserved, set to zero */

USHORT      rc;          /* return code */
```

## Assembler Language

```
EXTRN DosSetProcCp:FAR
INCL_DOSNLS EQU 1
```

```
PUSH WORD CodePage ;Code page identifier
PUSH WORD 0 ;Reserved (must be zero)
CALL DosSetProcCp
```

Returns WORD

# DosSetPrty – Set Process Priority

This call allows a process to change the priority of all the threads of any process; or all the threads of the current process or a child process, as well as any descendants; or a single thread within the current process. When a process changes the priority of threads in other processes, only default priorities are changed.

**DosSetPrty (Scope, PriorityClass, PriorityDelta, ID)**

## Parameters

**Scope (USHORT)** – input

The extent of the priority change.

Value	Definition
-------	------------

0	All the threads of any process.
---	---------------------------------

1	All the threads of a process and any descendants. The indicated process must be the current process or a process created by the current process. Detached processes may not be specified. The indicated process may possibly have terminated.
---	---

2	A single thread of the current process.
---	---

**PriorityClass (USHORT)** – input

Priority class of a process. The values and descriptions are:

Value	Definition
-------	------------

0	No change, leave as is
---	------------------------

1	Idle-time
---	-----------

2	Regular
---	---------

3	Time-critical.
---	----------------

4	Fixed-high.
---	-------------

**PriorityDelta (SHORT)** – input

Delta priority to apply to the process's current base priority level. This value must range from –31 to +31.

**ID (USHORT)** – input

A process ID (scope = 0 or 1) or a thread ID (scope = 2). If this operand is equal to zero, the current process or thread is assumed.

**rc (USHORT)** – return

Return code descriptions are:

0	NO_ERROR
---	----------

303	ERROR_INVALID_PROCID
-----	----------------------

304	ERROR_INVALID_PDELTA
-----	----------------------

305	ERROR_NOT_DESCENDANT
-----	----------------------

307	ERROR_INVALID_PCLASS
-----	----------------------

308	ERROR_INVALID_SCOPE
-----	---------------------

309	ERROR_INVALID_THREADID
-----	------------------------

## Remarks

The OS/2 scheduler has a concept of priority classes and levels. DosSetPrty allows threads to move between classes in response to changes in their execution environments. Within each class, a thread's priority level can vary because of a DosSetPrty request or action taken by the system. System-initiated priority variation is performed as a combination of a specific thread's actions and the overall system activity.

A time-critical thread has the highest priority class and executes before any fixed-high, regular, or idle-time threads. A fixed-high thread has a priority class that is lower than time-critical but executes before any regular or idle-time threads.



# DosSetPrty —

## Set Process Priority

Time-critical threads have static priorities that are not varied by OS/2. Threads are scheduled in priority order, with round-robin scheduling of threads of equal priority.

For each of the four priority classes, there are 32 distinct priority levels, 0 to +31. Whenever `DosSetPrty` is issued with a class specification, but no value is specified for `PriorityDelta`, the base priority level defaults to zero.

The priority level of a process consists of a computed priority value that is based upon the display status (foreground or background) of the process, its recent I/O and processor time-usage history, and other factors. The signed value specified in `PriorityDelta` is added to the computed priority to produce the actual priority used by the scheduler. The result is restricted to the valid range, based upon the current priority class.

Specifying a higher priority delta allows a thread to obtain better processor scheduling than it normally would. A lower priority delta gives the thread less processor resource than it normally receives.

When used with `PriorityClass` to change to a different class, the delta value applies to the base priority. A new base level of 0 is assigned the target thread and any `PriorityDelta` specified is relative to zero.

A process can change the priority of any thread within its current process. However, when a process changes the priority of threads in another process, only those with default priorities are changed. The priority of any thread in another process that has explicitly changed its priority from the default with `DosSetPrty` is not changed.

The initial thread of execution for an application is given a regular class priority that varies by the system. A thread started with `DosCreateThread` inherits the priority of the thread in the process that creates it. A child process started by a `DosExecPgm` call inherits the priority of the thread in the parent process that creates it.

## C Language

```
#define INCL_DOSPROCESS

USHORT rc = DosSetPrty(Scope, PriorityClass, PriorityDelta, ID);

USHORT      Scope;          /* Indicate scope of change */
USHORT      PriorityClass;  /* Priority class to set */
SHORT       PriorityDelta;  /* Priority delta to apply */
USHORT      ID;            /* Process or thread ID */

USHORT      rc;            /* return code */
```

## Assembler Language

```
EXTRN DosSetPrty:FAR
INCL_DOSPROCESS EQU 1

PUSH WORD Scope ;Indicate scope of change
PUSH WORD PriorityClass ;Priority class to set
PUSH WORD PriorityDelta ;Priority delta to apply
PUSH WORD ID ;Process or thread ID
CALL DosSetPrty
```

Returns WORD

## Example

The following example illustrates how to obtain the priority of a thread and how to change the priority. The main thread creates `Thread2` and allows it to begin executing. `Thread2` iterates through a loop that prints a line and then sleeps, relinquishing its time slice to the main thread. After one or two iterations by `Thread2`, the main thread obtains `Thread2`'s priority information and prints it. It then raises `Thread2`'s priority to fixed-high, and increments the level by ten. Since `Thread2` is now at a high priority, it immediately finishes its remaining iterations before relinquishing control on a long sleep; at this point, the main thread re-examines `Thread2`'s priority and reports its new priority level. In this example, it is

## DosSetPrty – Set Process Priority

helpful to understand how the DosSleep calls are used either to relinquish control of the processor, or to keep a thread alive (see DosTimerAsync or DosTimerStart for alternatives to DosSleep).

```
#define INCL_DOSPROCESS

#include <os2.h>

#define PRTYC_FIXEDHIGH 4      /* Priority class: fixed-high */
#define PRTY_DELTA 10         /* Priority delta: increase by 10 */
#define SEGSIZE 4000          /* Number of bytes requested in segment */
#define ALLOCFLAGS 0          /* Segment allocation flags - no sharing */
#define SLEEPSHORT 0L         /* Sleep interval - 5 milliseconds */
#define SLEEPLONG 20L         /* Sleep interval - 75 milliseconds */
#define RETURN_CODE 0         /* Return code for DosExit() */

VOID APIENTRY Thread2()
{
    USHORT i;

    /* Loop with four iterations */
    for(i=1; i<5; i++)
    {
        printf("In Thread2, i is now %d\n", i);

        /** Sleep to relinquish time slice to main thread **/
        DosSleep(SLEEPSHORT);      /* Sleep interval */
    }
    DosExit(EXIT_THREAD,          /* Action code - end a thread */
            RETURN_CODE);         /* Return code */
}
```

## DosSetPrtY — Set Process Priority

```
main()
{
    USHORT    Priority;        /* Thread priority */
    USHORT    Class;          /* Priority class */
    USHORT    Level;          /* Priority level */
    SEL        ThreadStackSel; /* Segment selector for thread stack */
    PBYTE     StackEnd;       /* Ptr. to end of thread stack */
    USHORT    rc;

    /* Allocate segment for thread stack; this is better than just */
    /* declaring an array of bytes to use as a stack. Make pointer eos. */
    rc = DosAllocSeg(SEGSIZE,          /* Number of bytes requested */
                    &ThreadStackSel, /* Segment selector (returned) */
                    ALLOCFLAGS);      /* Allocation flags */
    StackEnd = MAKEP(ThreadStackSel, SEGSIZE-1);

    /* Start Thread2 */
    if(!DosCreateThread((PFNTHREAD) Thread2, /* Thread address */
                      &ThreadID,           /* Thread ID (returned) */
                      StackEnd))           /* End of thread stack */
        printf("Thread2 created.\n");

    /** Sleep to allow Thread2 to execute **/
    if(!DosSleep(SLEEPLONG))              /* Sleep interval */
        printf("Slept a little to let Thread2 execute.\n");

    /** Obtain Thread2's priority information and report it **/
    if(!rc=DosGetPrtY(PRTYS_THREAD,      /* Scope - single thread */
                    &Priority,           /* Address to put priority */
                    ThreadID))           /* ID - thread ID */
    {
        /* Extract priority class and level information */
        Class = HIBYTE(Priority);
        Level = LOBYTE(Priority);
        printf("Thread2: ID is %d, Priority Class is %d and Level is %d\n",
              ThreadID, Class, Level);
    }

    /** Raise Thread2's priority **/
    if(!rc=DosSetPrtY(PRTYS_THREAD,      /* Scope - single thread */
                    PRTYC_FIXEDHIGH,     /* Prty class - fixed-high */
                    PRTY_DELTA,          /* Prty delta - increase by 10 */
                    ThreadID))           /* ID - thread ID */
    {
        /* Obtain Thread2' new priority information and report it */
        rc=DosGetPrtY(PRTYS_THREAD,      /* Scope - single thread */
                    &Priority,           /* Address to put priority */
                    ThreadID);           /* ID - thread ID */

        /* Extract priority class and level information */
        Class = HIBYTE(Priority);
        Level = LOBYTE(Priority);
        printf("Thread2: ID is %d, New Priority Class is %d and Level is %d\n",
              ThreadID, Class, Level);
    }
}
```

This call sets the status of a child session.

<b>DosSetSession</b> (SessID, StatusData)
---

Parameters

- SessID** (*USHORT*) – input  
ID of the target session. The value specified for SessID must have been returned on a prior call to DosStartSession.
- StatusData** (*PSTATUSDATA*) – input  
Address of the session status data structure:
- length** (*USHORT*)  
Length of structure, including length.
- 6**            Only valid value.
- selectInd** (*USHORT*)  
Specifies whether the target session is flagged as selectable or non-selectable: The operator can continue to select a non-selectable (bonded) windowed session by pressing a mouse button within a visible part of the window.
- | Value | Definition                       |
|-------|----------------------------------|
| 0     | Leave current setting unchanged. |
| 1     | Selectable.                      |
| 2     | Non-selectable.                  |
- bondInd** (*USHORT*)  
Specifies which session to bring to the foreground the next time the parent session is selected. The operator may continue to select a non-selectable (bonded) windowed session by pressing a mouse button within a visible part of the window.
- | Value | Definition   |
|-------|--|
| 0     | Leave current setting unchanged.   |
| 1     | Establish a bond between parent session and child session. The child session is brought to the foreground the next time the parent session is selected, or when the child session itself is selected.                  |
| 2     | Bring either the parent session to the foreground the next time the parent session is selected, or the child session if the child session is selected. Any bond previously established with a child session is broken. |
- rc** (*USHORT*) – return  
Return code descriptions are:
- |     |                               |
|-----|-------------------------------|
| 0   | NO_ERROR                      |
| 369 | ERROR_SMG_INVALID_SESSION_ID  |
| 418 | ERROR_SMG_INVALID_CALL        |
| 452 | ERROR_SMG_SESSION_NOT_PARENT  |
| 455 | ERROR_SMG_INVALID_BOND_OPTION |
| 456 | ERROR_SMG_INVALID_SELECT_OPT  |
| 461 | ERROR_SMG_INVALID_DATA_LENGTH |

# DosSetSession —

## Set Session Status

### Remarks

DosSetSession sets one or both of the following structure elements related to a child session. The elements can be set individually by the parent session, and either one can be changed without affecting the current setting of the other:

- selectInd** Sets the child session selectable or non-selectable.
- bondInd** Bonds the child session to the parent session. If the operator selects the parent session from the Task Manager, the child session is brought to the foreground.

These elements only affect selections made by the operator from the switch list, not selections made by the parent session. When a parent session selects its own session, the parent session is brought to the foreground even if a bond is in effect. When a parent session selects a child session, the child session is brought to the foreground even if the parent session had set the child session to be non-selectable.

DosSetSession may be issued by a process only for a child session it started with a DosStartSession request, specifying Related = 1. Neither the parent session nor any grandchild session may be the target of DosSetSession.

A bond established between a parent session and a child session can be broken by reissuing DosSetSession and specifying either:

- bondInd = 2** Breaks the bond between the parent session and the child session.
- bondInd = 1** Establishes a bond with a different child session. In this case the bond with the previous child session is broken.

Assume a bond is established between session A and its immediate child session B. Assume another bond is established between session B and its immediate child session C. Now if the operator selects session A, session C is brought to the foreground. However, if session A selects its own session, session A is brought to the foreground. If session A selects session B, session C is brought to the foreground. In the latter case, the bond between B and C is honored.

Assume a bond is established between session A and its immediate child session B, and assume B is non-selectable. The operator cannot select session B directly. However, if the operator selects session A, session B is brought to the foreground.

A parent session can run in either the foreground or background when DosSetSession is issued.

### C Language

```
typedef struct _STATUSDATA { /* stsdta */

    USHORT Length;           /* length of this data structure */
    USHORT SelectInd;        /* 0=leave setting unchanged, 1=selectable
                             2=non-selectable */
    USHORT BondInd;          /* which session to bring to foreground */

} STATUSDATA;

#define INCL_DOSSESMGR

USHORT rc = DosSetSession(SessID, StatusData);

USHORT SessID; /* Session ID */
PSTATUSDATA StatusData; /* Session status data */

USHORT rc; /* return code */
```

## DosSetSession — Set Session Status

### Assembler Language

```
STATUSDATA struc

    stsddata_Length    dw ? ;length of this data structure
    stsddata_SelectInd dw ? ;0=leave setting unchanged, 1=selectable
                           ;2=non-selectable
    stsddata_BindInd   dw ? ;which session to bring to foreground

STATUSDATA ends

EXTRN DosSetSession:FAR
INCL_DOSSESMGR      EQU 1

PUSH  WORD    SessID      ;Session ID
PUSH@ OTHER    StatusData ;Session status data
CALL  DosSetSession

Returns WORD
```

# DosSetSigHandler – Set Signal Handler

FAPI

This call notifies OS/2 of a handler for a signal. It may also be used to ignore a signal or install a default action for a signal.

**DosSetSigHandler** (Routine, PrevAddress, PrevAction, Action, SigNumber)

## Parameters

**Routine** (*PFNSIGHANDLER*) – input

Address of the entry point of routine that receives control when a signal equal to SigNumber is received.

**PrevAddress** (*PFNSIGHANDLER FAR \**) – output

Address of the previous signal handler. This operand may be coded as null (= 0), then it is ignored.

**PrevAction** (*PUSHORT*) – output

Address of the previous signal action. Only values 0 to 3 are returned. This operand may be coded as null (= 0), then it is ignored.

**Action** (*USHORT*) – input

Indicates what action to take when the specified signal is received:

Value	Definition
0	The system default action is installed for the signal.
1	The signal is to be ignored.
2	The routine receives control when the SigNumber occurs.
3	It is an error for any program to signal this SigNumber to this process.
4	The current signal is reset without affecting the disposition of the signal.

**SigNumber** (*USHORT*) – input

Signal number to be intercepted by this signal handler. The signal numbers defined are:

Value	Definition
1	Ctrl-C (SIGINTR)
3	Program terminated (SIGTERM)
4	Ctrl-Break (SIGBREAK)
5	Process flag A (SIGPFA)
6	Process flag B (SIGPFB)
7	Process flag C (SIGPFC)

**Note:** Presentation Manager applications may not establish signal handlers for Ctrl-C and Ctrl-Break. Establishing a signal handler for Ctrl-C and Ctrl-Break is supported for VIO-Windowable and full-screen applications.

The following chart indicates what signal to specify to cause the signal handler to get control for the CTRL-C and CTRL-Break key sequences in each of the keyboard modes (ASCII and Binary):

	ASCII Mode	BINARY Mode
CTRL-C	SIGINTR	
CTRL-Break	SIGINTR	SIGBREAK

**rc** (*USHORT*) – return

Return code descriptions are:

0	NO_ERROR
1	ERROR_INVALID_FUNCTION
209	ERROR_INVALID_SIGNAL_NUMBER
210	ERROR_THREAD_1_INACTIVE

## Remarks

When the signal indicated by SigNumber occurs, the signal handling routine receives control with:

(SS:SP) Far return address

(SS:SP + 4) SigNumber being processed

(SS:SP + 6) SigArg furnished on the DosFlagProcess request, if appropriate.

Other than SS, SP, CS, IP and flags, all registers contain the same values they contained at the time the signal was received. The handler may exit by executing an intersegment return instruction, or by manually setting the stack frame to some known state and jumping to some known location. If the former option is selected, execution resumes where it was interrupted, and all registers are restored to their values at the time of the interruption.

The signal handler is given control under the first thread of a process, not a thread created by the DosCreateThread system request. It is invalid to issue this system call when thread 1 has terminated. If thread 1 terminates with other threads still active, all signals are reset to the default action.

To return from the signal, the handler must remove the signal number and signal argument passed as parameters. For handlers written in most high-level languages, this is done automatically. A handler written in assembler language must execute a Far RET 4 instruction or its equivalent, to return to the caller. The signal handler may also reset the stack pointer to some previous valid stack frame and jump to some other part of the program.

The values returned in PrevAddress and PrevAction are to be used for restoring the previous signal handler when the current process no longer wishes to intercept this signal. For Action = 4, no values are returned for PrevAddress or PrevAction.

When a signal is issued from the base keyboard device driver in response to a Ctrl-C or Ctrl-Break key press, the default action terminates the process if the application did not install a signal handler for any signal numbers 1-4.

For signals of type SIGINTR or SIGBREAK, a call to DosSetSigHandler also determines which process within the current session is signalled as a result of a device driver call to Device Helper Services for the SendEvent function and CTRL-C (or CTRL-BREAK) event type. (See the *IBM Operating System/2 Version 1.2 I/O Subsystems And Device Support Volume 1*, Device Helper Services discussion). This process is known as the 'signal focus' for SIGINTR (or SIGBREAK) within its session. The signal focus for SIGINTR need not be the same process as the signal focus for SIGBREAK. The determination for signal focus follows.

Initially, a session has no signal focus for SIGINTR (or SIGBREAK). A process becomes the signal focus for SIGINTR (or SIGBREAK) within its session if it calls DosSetSigHandler with ActionCode equal to 1, 2, or 3. A process remains the signal focus until:

- The process terminates.
- The process calls DosSetSigHandler with ActionCode equal to zero.
- Another process calls DosSetSigHandler with ActionCode equal to 1, 2, or 3.

In the first two cases, the parent or its closest related ancestor process that has a handler installed for the appropriate signal becomes the focus. If no eligible process exists, the session ceases to have a signal focus for that signal.

If a device driver makes a SendEvent call for CTRL-C or CTRL-BREAK and the current session has no focus for the corresponding signal, all processes in the session are signaled with SIGTERM to terminate.



# DosSetSigHandler — Set Signal Handler

FAPi

## Family API Considerations

Some options operate differently in the DOS mode than in OS/2 mode. Therefore, the following restriction applies to DosSetSigHandler when coding in DOS mode:

- The only signals recognized in DOS are SIGINTR (Ctrl-C) and SIGBREAK.
- The option Action=3 generates an "invalid signal number" error.
- If SigNumber is any value other than SIGINTR or SIGBREAK, then an "invalid signal number" error is generated.

SIGINTR is fully supported, and SIGBREAK is related to SIGINTR. Therefore, if SIGINTR is specified, both SIGINTR and SIGBREAK are transferred to the SIGINTR handler. SIGBREAK is permitted as a coded value, but the request to set SIGBREAK is ignored. To be compatible in all environments, SIGBREAK and SIGINTR should be considered together in all cases.

## C Language

```
#define INCL_DOSSIGNALS
```

```
USHORT rc = DosSetSigHandler(Routine, PrevAddress, PrevAction, Action,  
                             SigNumber);
```

```
PFNSIGHANDLER Routine; /* Signal handler */  
PFNSIGHANDLER FAR * PrevAddress; /* Previous handler (returned) */  
USHORT PrevAction; /* Previous action (returned) */  
USHORT Action; /* Indicate request type */  
USHORT SigNumber; /* Signal number of interest */  
  
USHORT rc; /* return code */
```

## Assembler Language

```
EXTRN DosSetSigHandler:FAR  
INCL_DOSSIGNALS EQU 1
```

```
PUSH@ DWORD Routine ;Signal handler  
PUSH@ DWORD PrevAddress ;Previous handler (returned)  
PUSH@ WORD PrevAction ;Previous action (returned)  
PUSH WORD Action ;Indicate request type  
PUSH WORD SigNumber ;Signal number of interest  
CALL DosSetSigHandler
```

Returns WORD

## Example

The following example illustrates the use of a user-defined flag to signal time-critical events. The main thread installs a routine, named FlagA\_Handler(), as the signal handler for user-defined Flag A. It then creates a thread and blocks on a reserved RAM semaphore; this thread obtains its process ID and signals the main thread via Flag A. The main thread responds by executing the signal handler.

```
#define INCL_DOSPROCESS
#define INCL_DOSIGNALS
#define INCL_DOSERRORS

#include <os2.h>

#define TIMEOUT          5000L

TID          ThreadID;
BYTE         ThreadStack[4000];

VOID APIENTRY FlagA_Handler(arg1, arg2)      /* Define signal handler */
{
    USHORT    arg1;
    USHORT    arg2;
    {
        printf("Handler for Flag A now running.\n");
        return;
    }
}

VOID APIENTRY Thread_A()
{
    PIDINFO    PidInfo;
    USHORT     FlagArg;
    USHORT     rc;

    DosGetPID(&PidInfo);
    printf("Process ID is %d\n", PidInfo.pid);
    if(!rc = DosFlagProcess(PidInfo.pid,
                           FLGP_PID,
                           PFLG_A,
                           FlagArg))
        printf("FlagA signal sent from ThreadA to main thread.\n");
    else
        printf("FlagProcess rc is %d\n", rc)/* Error processing on rc */;
    DosExit(EXIT_THREAD,      /* Action Code */
            RETURN_CODE);      /* Result Code */
}

main()
{
    ULONG      RamSem = 0L;
    ULONG far  *RamSemHandle = &RamSem;
    USHORT     rc;

    if(!rc=DosSetSigHandler((PFNSIGHANDLER) FlagA_Handler,
                           NULL,
                           NULL,
                           SIGA_ACCEPT,
                           SIG_PFLG_A))
        printf("Main thread has set FlagA handler.\n");
    else
        /* Error processing on rc */;
    if(!rc=DosSemRequest(RamSemHandle,
                        TIMEOUT))
        printf("Semaphore obtained.\n");
    if(!(DosCreateThread((PFNTHREAD) Thread_A,
                        &ThreadID,
                        &ThreadStack[3999])))
        printf("ThreadA created.\n");
    printf("Main thread will now wait on a Ramsem for a while.\n");
    if((rc=DosSemRequest(RamSemHandle,
                        TIMEOUT))
        == ERROR_INTERRUPT)
        printf("Main thread interrupted while waiting, rc is %d.\n", rc);
}
```

# DosSetVec —

## Establish Handler for Exception Vector

FAPI

This call allows a process to register an address to be used when a machine exception occurs.

**DosSetVec (VecNum, Routine, PrevAddress)**

### Parameters

**VecNum** (*USHORT*) — input

Number of the vector to be serviced by this routine. Valid values are:

Value	Definition
00	Divide overflow
04	Overflow
05	Bound
06	Invalid opcode
07	Processor extension not available
16	Processor extension error.

**Routine** (*PFN*) — input

Address of a routine to be entered when the exception occurs. If this parameter is 0, any previous address is de-registered.

**PrevAddress** (*PFN*) — output

Address of the previous handler routine. This is provided so a handler may be set, then later restored to the previous handler.

**rc** (*USHORT*) — return

Return code descriptions are:

1	ERROR_INVALID_FUNCTION
50	ERROR_NOT_SUPPORTED

### Remarks

The DosSetVec process is analogous to setting an address in the interrupt vector table when running in 8086 mode.

Should an exception occur, and the process has registered a handler, that handler is entered as if its address had been stored in the CPU's interrupt vector, except that the interrupt is still enabled. If no address has been registered for that vector, the process terminates.

When a process registers an exception handler for VecNum 7 (processor extension not available), the machine status word (MSW) for that process is set to indicate that a numeric processor extension (NPX) 287 is not present in the machine. The Emulate bit is set and the Monitor Processor bit is reset. This is done without regard for the true state of the hardware.

When a process de-registers a handler for VecNum 7, the MSW is set to reflect the true state of the hardware.

When an NPX287 exception is processing, the NPX287 status word is passed to the exception handler by being pushed on the stack before the exception handler is invoked. When the exception handler has completed execution, this word must be popped from the stack before an IRET is issued to return to the exception handler interface routine.

### Family API Considerations

Some options operate differently in the DOS mode than in OS/2 mode. Therefore, the following restriction applies to DosSetVec when coding for the DOS mode.

VecNum = 7 not supported.

# DosSetVec — Establish Handler for Exception Vector

## C Language

```
#define INCL_DOSMISC

USHORT rc = DosSetVec(VecNum, Routine, PrevAddress);

USHORT      VecNum;          /* Function request code */
PFN          Routine;        /* Handler routine */
PFN          PrevAddress;     /* Previous handler address (returned) */

USHORT      rc;              /* return code */
```

## Assembler Language

```
EXTRN DosSetVec:FAR
INCL_DOSMISC      EQU 1

PUSH WORD VecNum      ;Function request code
PUSH@ OTHER Routine   ;Handler routine address
PUSH@ DWORD PrevAddress ;Previous handler address (returned)
CALL DosSetVec

Returns WORD
```

# DosSetVerify — Set/Reset Verify Switch

FAP1

---

This call sets the verify switch.

DosSetVerify (VerifySetting)

## Parameters

**VerifySetting** (*USHORT*) — input

New state of Verify Mode for the requesting process as shown below:

Value	Definition
0	Verify mode is not active.
1	Verify mode is active.

**rc** (*USHORT*) — return

Return code descriptions are:

0	NO_ERROR
118	ERROR_INVALID_VERIFY_SWITCH

## Remarks

When verify is active, OS/2 performs a verify operation each time it does a file write to assure proper data recording on the disk. Although disk recording errors are rare, this function has been provided for applications to verify the proper recording of critical data.

## C Language

```
#define INCL_DOSFILEMGR

USHORT rc = DosSetVerify(VerifySetting);

USHORT VerifySetting; /* New value of verify switch */

USHORT rc; /* return code */
```

## Assembler Language

```
EXTRN DosSetVerify:FAR
INCL_DOSFILEMGR EQU 1

PUSH WORD VerifySetting ;New value of verify switch
CALL DosSetVerify
```

Returns WORD

This call returns the size of a segment.

**DosSizeSeg (Selector, Size)**

## Parameters

**Selector (SEL)** — input

Selector/segment of the segment for which size is to be determined. This must be the base selector in the case of a huge segment.

**Size (PULONG)** — output

Address of the returned segment size. For non-huge segments, size is in the range 0 through 64KB. For huge segments, size equals (NUMSEG < 16) + ASIZE where NUMSEG and ASIZE are the last values passed successfully to DosAllocHuge or DosReallocHuge for this huge segment.

**rc (USHORT)** — return

Return code description is:

0            NO\_ERROR

## Remarks

This function provides compatibility for family applications that must run in DOS or OS/2 mode.

DosSizeSeg returns the actual size of memory allocated by a DosAllocSeg or DosAllocHuge request and is intended for use when the application is running in DOS mode, where allocations are rounded up to the next paragraph.

## C Language

```
#define INCL_DOSMEMMGR
```

```
USHORT rc = DosSizeSeg(Selector, Size);
```

```
SEL          Selector;      /* Selector/Segment */
PULONG       Size;          /* Size of segment (returned) */

USHORT       rc;            /* return code */
```

## Assembler Language

```
EXTRN DosSizeSeg:FAR
INCL_DOSMEMMGR EQU 1

PUSH WORD Selector      ;Selector/Segment
PUSH0 DWORD Size        ;Size of segment (returned)
CALL DosSizeSeg
```

Returns WORD

# DosShutdown — Shutdown File Systems for Power Off

This call locks out changes to all file systems and forces system buffers to disk in preparation for system power off.

<b>DosSizeSeg (Reserved)</b>
------------------------------

## Parameters

**Reserved** (*ULONG*) — input

Double-word whose value must be zero.

**rc** (*USHORT*) — return

Return code description is:

0	NO_ERROR
274	ERROR_ALREADY_SHUTDOWN
87	ERROR_INVALID_PARAMETER

## Remarks

DosShutdown may take several minutes to complete depending on the amount of data buffered.

Other API function calls that change file system data called while the system is shutdown will either return the error ERROR\_ALREADY\_SHUTDOWN or block permanently.

It should be noted that it will not be possible to increase memory overcommit once the DosShutdown has been called. This means that in low memory situations some functions may fail due to a lack of memory. This is of particular importance to the process calling DosShutdown. All memory that the calling process will ever need should be allocated before calling DosShutdown. This includes implicit memory allocation that may be done on behalf of the caller by system functions.

When DosShutdown returns successfully, it is safe to power the system off or to reboot it.

## C Language

```
#define INCL_DOSFILEMGR

USHORT rc = DosShutdown(Reserved);

ULONG      Reserved;      /* Reserved, must be set to zero */

USHORT      rc;           /* return code */
```

## Assembler Language

```
EXTRN DosShutdown:FAR
INCL_DOSFILEMGR EQU 1

PUSH DWORD Reserved      ;Reserved, must be set to zero
CALL DosShutdown

Returns WORD
```

---

This call suspends the current thread for a specified time. If the requested interval is 0, the call gives up the remainder of the current time slice.

**DosSleep (TimeInterval)**

## Parameters

**TimeInterval** (*ULONG*) – input

Time interval in milliseconds until the thread is awakened.

**rc** (*USHORT*) – return

Return code descriptions are:

<b>0</b>	<b>NO_ERROR</b>
<b>322</b>	<b>ERROR_TS_WAKEUP</b>

## Remarks

DosSleep suspends the current thread for the specified time period. The actual time it is asleep may be off by a clock tick or two, depending on the execution status of other threads running in the system.

If the time is 0, the thread gives up the remainder of the current time slice and allows any other ready threads of equal priority to run with the current thread for its next slice. Because the amount of sleep time specified is 0, an immediate return with 0 delay is made if no other ready thread is found. However, DosSleep does not yield to a thread of lower priority.

If the time is non-0, the time is rounded up to the resolution of the scheduler clock.

If DosSleep is used to regularly poll an external source to determine the occurrence of some event, a time equal to the longest response interval should be used.

For short time intervals, the rounding-up process combined with the thread priority interactions may cause a sleeping interval to be longer than requested. Also, when a process completes sleeping, it is scheduled for execution. But that execution could be delayed by hardware interrupts or by another thread running at a higher priority. A program should not use the DosSleep call as a substitute for a real-time clock because rounding of the sleep interval causes cumulative errors.

Asynchronous timers can be started with DosTimerAsync and DosTimerStart. DosTimerAsync starts a one-shot asynchronous timer, and DosTimerStart starts a periodic interval timer. DosTimerStop is issued to stop these timers.

**Note:** To ensure optimum performance, you should not use DosSleep in a single-thread Presentation Manager application. See WinStartTimer.

## Family API Considerations

Some options operate differently in the DOS mode than in OS/2 mode. Therefore, the following restrictions apply to DosSleep when coding in DOS mode:

- DosSleep accuracy can be in error by 0.5%.
- DosSleep can degrade system performance of non-foreground program operations when DOS mode is in foreground.



# DosSleep — Delay Process Execution

FAPI

## C Language

```
#define INCL_DOSPROCESS

USHORT rc = DosSleep(TimeInterval);

ULONG      TimeInterval; /* Interval size (in milliseconds) */

USHORT      rc;          /* return code */
```

## Assembler Language

```
EXTRN DosSleep:FAR
INCL_DOSPROCESS EQU 1

PUSH  DWORD  TimeInterval ;Interval size (in milliseconds)
CALL  DosSleep

Returns WORD
```

## Example

The following example illustrates how to obtain the priority of a thread and how to change the priority. The main thread creates Thread2 and allows it to begin executing. Thread2 iterates through a loop that prints a line and then sleeps, relinquishing its time slice to the main thread. After one or two iterations by Thread2, the main thread obtains Thread2's priority information and prints it. It then raises Thread2's priority to fixed-high, and increments the level by ten. Since Thread2 is now at a high priority, it immediately finishes its remaining iterations before relinquishing control on a long sleep; at this point, the main thread re-examines Thread2's priority and reports its new priority level. In this example, it is helpful to understand how the DosSleep calls are used either to relinquish control of the processor, or to keep a thread alive (see DosTimerAsync or DosTimerStart for alternatives to DosSleep).

```
#define INCL_DOSPROCESS

#include <os2.h>

#define PRTYC_FIXEDHIGH 4 /* Priority class: fixed-high */
#define PRTY_DELTA 10 /* Priority delta: increase by 10 */
#define SEGSIZE 4000 /* Number of bytes requested in segment */
#define ALLOCFLAGS 0 /* Segment allocation flags - no sharing */
#define SLEEPSHORT 0L /* Sleep interval - 5 milliseconds */
#define SLEEPLONG 20L /* Sleep interval - 75 milliseconds */
#define RETURN_CODE 0 /* Return code for DosExit() */

VOID APIENTRY Thread2()
{
    USHORT i;

    /* Loop with four iterations */
    for(i=1; i<5; i++)
    {
        printf("In Thread2, i is now %d\n", i);

        /** Sleep to relinquish time slice to main thread **/
        DosSleep(SLEEPSHORT); /* Sleep interval */
    }
    DosExit(EXIT_THREAD, /* Action code - end a thread */
            RETURN_CODE); /* Return code */
}
```

```
main()
{
    USHORT    Priority;           /* Thread priority */
    USHORT    Class;             /* Priority class */
    USHORT    Level;             /* Priority level */
    SEL        ThreadStackSel;   /* Segment selector for thread stack */
    PBYTE     StackEnd;          /* Ptr. to end of thread stack */
    USHORT    rc;

    /* Allocate segment for thread stack; this is better than just */
    /* declaring an array of bytes to use as a stack. Make pointer eos. */
    rc = DosAllocSeg(SEGSIZE,           /* Number of bytes requested */
                    &ThreadStackSel,   /* Segment selector (returned) */
                    ALLOCFLAGS);       /* Allocation flags */
    StackEnd = MAKEP(ThreadStackSel, SEGSIZE-1);

    /* Start Thread2 */
    if(!DosCreateThread((PFNTHREAD) Thread2, /* Thread address */
                       &ThreadID,          /* Thread ID (returned) */
                       StackEnd))           /* End of thread stack */
        printf("Thread2 created.\n");

    /** Sleep to allow Thread2 to execute **/
    if(!DosSleep(SLEEPLONG))              /* Sleep interval */
        printf("Slept a little to let Thread2 execute.\n");

    /** Obtain Thread2's priority information and report it **/
    if(!rc=DosGetPrty(PRTYS_THREAD,        /* Scope - single thread */
                     &Priority,            /* Address to put priority */
                     ThreadID))            /* ID - thread ID */
    {
        /* Extract priority class and level information */
        Class = HIBYTE(Priority);
        Level = LOBYTE(Priority);
        printf("Thread2: ID is %d, Priority Class is %d and Level is %d\n",
              ThreadID, Class, Level);
    }

    /** Raise Thread2's priority **/
    if(!rc=DosSetPrty(PRTYS_THREAD,        /* Scope - single thread */
                     PRTYC_FIXEDHIGH,      /* Prty class - fixed-high */
                     PRTY_DELTA,           /* Prty delta - increase by 10 */
                     ThreadID))            /* ID - thread ID */
    {
        /* Obtain Thread2' new priority information and report it */
        rc=DosGetPrty(PRTYS_THREAD,        /* Scope - single thread */
                     &Priority,            /* Address to put priority */
                     ThreadID);            /* ID - thread ID */

        /* Extract priority class and level information */
        Class = HIBYTE(Priority);
        Level = LOBYTE(Priority);
        printf("Thread2: ID is %d, New Priority Class is %d and Level is %d\n",
              ThreadID, Class, Level);
    }
}
```

# DosSMRegisterDD – Register Session Switch Notification

Allows a device driver to register itself with the Session Manager.

DosSMRegisterDD (RegisterData)

## Parameters

**RegisterData** (*PREGISTERDATA*) – input  
Address of the structure containing the RegisterData necessary for this call.

**Length** (*USHORT*)  
Length of the data structure in bytes including Length itself. Length is 8 for OS/2 version 1.2.

**Notification** (*USHORT*)  
Bit map that informs the session manager when to issue the notification IOCTL to the device driver. The bits of this word are defined as follows:

Bit	Description
15 – 4	Reserved and must be set to zero.
3	Session termination notification. Action = 8
2	Post-Session restore notification. Action = 4
1	Post-Session save notification. Action = 2
0	Pre-Session save notification. Action = 1

Action is the unsigned integer in the Action field of the notification IOCTL. This tells the device driver what is happening. Notice that the integers correspond to bits 0 to 3 being set.

**DDName** (*ULONG*)  
Address of the device name, as an ASCII string (ie. SCREEN\$, KBD\$, etc.).

**rc** (*USHORT*) – return  
Return code descriptions are:

418	ERROR_SMG_INVALID_CALL
461	ERROR_SMG_INVALID_DATA_LENGTH
515	ERROR_SMG_TOO_MANY_DDS
516	ERROR_SMG_INVALID_NOTIFICATION

## Remarks

The Session Manger calls the registered device driver during a session switch, based on the specific form of notifications required.

The notification IOCTL passed to the device driver has the following format:

Size	Definition
<b>WORD</b>	DevIOCtl packet length, including the length itself
<b>WORD</b>	Screen switch notification type
<b>WORD</b>	Incoming session number
<b>WORD</b>	Outgoing session number.

The device driver passes a bit map that informs the session manager when the device driver needs to be called. The possible phases are:

- Before a session save
- After a session save (before a restore)
- After a session restore
- After a session termination.

The device drivers must issue this API only during their initialization phase. After the session manager has been initialized, it rejects all further calls to this API, returning ERROR\_SMG\_INVALID\_CALL.

## DosSMRegisterDD — Register Session Switch Notification

### C Language

```
#define INCL_DOSFILEMGR

USHORT rc = DosSMRegisterDD(RegisterData);

PREREGISTERDATA RegisterData;      /* Data packet */

USHORT rc;                          /* return code */
```

### Assembler Language

```
EXTRN DosSMRegisterDD:FAR
INCL_DOSFILEMGR EQU 1

PUSH@ OTHER RegisterData ;Data packet
CALL DosSMRegisterDD

Returns WORD
```

# DosStartSession —

## Start Session

---

This call allows a program to start another program in a session.

DosStartSession (StartData, SessID, ProclD)

### Parameters

**StartData** (*PSTARTDATA*) — input

Address of the start session structure:

**length** (*USHORT*)

Length in bytes of the data structure, including length. A length of 24 bytes may be specified for OS/2 1.0 applications, or for applications that do not take advantage of the environment or windowing data.

A length of 30 bytes may be specified for OS/2 applications that want to use only the environment and inheritance features.

A length of 50 bytes may be selected for applications that want to use all the functions provided by DosStartSession. However, a length of 50 bytes is not allowed if the Session Manager detects that the Presentation Manager is not present.

**related** (*USHORT*)

Specifies whether the session created is related to the calling session, with the following values:

Value	Definition
0	New session is an independent session (not related).
1	New session is a child session (related).

An independent session is not a child session and cannot be controlled by the calling program. It cannot be specified as the target of DosSelectSession, DosSetSession, or DosStopSession. Note that termq is ignored for independent sessions, and the value of zero is returned for the SessID and ProclD parameters.

The calling program (parent session) may specify a child session as the target of DosSelectSession, DosSetSession, and DosStopSession, for related sessions. The SessID and ProclD parameters, and termq, are applicable only when related = 1 is specified.

Note also that for related sessions, although a parent session/child session relationship is established, a parent process/child process relationship is not established.

**fgbg** (*USHORT*)

Specifies whether the new session should be started in the foreground or background:

Value	Definition
0	Start session in foreground.
1	Start session in background.

**traceopt** (*USHORT*)

Specifies tracing options for the program started in the new session:

Value	Definition
0	Trace off.
1	Trace on; no notification of descendants.
2	Trace on; all descendant sessions.

For traceopt = 2, a termination queue must be supplied and related set to 1. Refer to "Debugger Considerations" in the Remarks section for additional information.

**pgmtitle** (*PSZ*)

A far pointer to an ASCIIZ string containing the program title. The string can be up to 61 bytes long, including the terminating byte of 0. If pgmtitle is zero or a NULL pointer, then DosStartSession defaults the program title to the name pgmname points to, minus any leading drive and path information.

# DosStartSession – Start Session

## **pgmname (PSZ)**

Can be one of the following:

- A far address to a NULL string.
- A NULL pointer (a far address equal to zero).
- An address of an ASCIIZ string containing the fully qualified drive, path, and file name of the program to be loaded.

If **pgmname** is a far address to a NULL string, or if it is a NULL pointer, the program name defaults to the command processor defined in the CONFIG.SYS/PROTSHELL statement. For example, if PROTSHELL = PMSHELL.EXE CMD.EXE, the program name defaults to CMD.EXE.

## **pgminputs (PBYTE)**

A far pointer to an ASCIIZ string containing the input arguments to be passed to the program.

**Note:** The maximum value allowed for the combined length of **pgmname** and **pgminputs** is 1024 characters. For more information on **pgmname** and **pgminputs**, see "Program Name/Program Input Considerations:" in the Remarks section.

## **termq (PBYTE)**

Can be one of the following:

- A far address to a NULL string.
- A NULL pointer (a far address equal to zero).
- A far pointer to an ASCIIZ string containing the fully qualified path and file name of an OS/2 queue created by a DosCreateQueue request. See "Parent/Child Termination Considerations" in the Remarks section for additional information.

## **environment (PBYTE)**

Can be one of the following:

- A far address to a NULL string.
- A NULL pointer (a far address equal to zero).
- A far pointer to an ASCIIZ environment string (see DosExecPgm) to be passed to the program started in the new session; it may be used for independent or related DosStartSession calls.

When **environment** = 0 (content of the string is not specified and a far address of 0 is passed), the program in the new session inherits the environment of the Shell if **inheritopt** = 0, or the environment of the program issuing the DosStartSession call if **inheritopt** = 1.

## **inheritopt (USHORT)**

Specifies whether the program started in the new session should inherit the calling process' environment and open file handles:

Value	Definition
0	Inherit from Shell process
1	Inherit from calling process.

Note that **inheritopt** is applicable whether the session being started is an independent or related session. After the DosStartSession request has completed, the new program's parent process is the Shell, not the process that issued the DosStartSession call. See "Parent/Child Relationships" in the Remarks section.

## **sessiontype (USHORT)**

Type of session that should be created for this program:

Value	Definition
0	Use <b>pgmhandle</b> or allow the Shell to establish the session type
1	Full screen session
2	Text-windowed session for programs using the Base Video Subsystem
3	Presentation Manager session.

## **iconfile (PSZ)**

Can be one of the following:

- A far address to a NULL string
- A NULL pointer (a far address equal to zero)

# DosStartSession —

## Start Session

- A far pointer to an ASCIIZ string containing the fully qualified drive, path and file name of an icon definition.

The system provides an icon for windowed applications if an icon file name is not provided on the DosStartSession call.

### **pgmhandle (ULONG)**

This is either 0 or the program handle returned by the WinAddProgram call. The program handle identifies the program in the installation file to be started, the program title, the session type, and the initial window size and position. However, information may be specified on the DosStartSession call to override the information in the installation file for this invocation of the program.

See "Program Handle Considerations:" in the Remarks section for more information.

### **pgmcontrol (USHORT)**

Specifies the initial state for a windowed application. This parameter is ignored for full-screen sessions. The initial window state may be defined as a combination of the following bit values:

Bit	Description
15	Use specified position and size.
14 – 4	Reserved and must be set to zero.
3	No auto close (for windowed session only).
2	Minimize.
1	Maximize.
0	Invisible.

### **initxpos (USHORT)**

Initial X coordinate in pels for the initial session window. Coordinates (0,0) indicate the bottom left corner of the display. This parameter is ignored for full-screen sessions.

### **initypos (USHORT)**

Initial Y coordinate in pels for the initial session window. Coordinates (0,0) indicate the bottom left corner of the display. It is ignored for full-screen sessions.

### **initxsize (USHORT)**

Initial X extent in pels for the initial session window. It is ignored for full-screen sessions.

### **initsize (USHORT)**

Initial Y extent in pels for the initial session window. It is ignored for full-screen sessions.

### **SessID (PUSHORT) – output**

Address of the session ID associated with the child session created. SessID is returned only when the value specified for related is 1. The SessID returned can be specified on subsequent calls to DosSelectSession, DosSetSession, and DosStopSession.

### **ProcID (PUSHORT) – output**

Address of the process ID associated with the child process created. ProcID is returned only when the value specified for related is 1. The ProcID returned may not be used on any OS/2 calls, for example, DosSetPrtty, that require a parent process/child process relationship. See "Parent/Child Relationships" in the Remarks section.

### **rc (USHORT) – return**

Return code descriptions are:

0	NO_ERROR
370	ERROR_SMG_NO_SESSIONS
418	ERROR_SMG_INVALID_CALL
453	ERROR_SMG_INVALID_START_MODE
454	ERROR_SMG_INVALID_RELATED_OPT
457	ERROR_SMG_START_IN_BACKGROUND
460	ERROR_SMG_PROCESS_NOT_PARENT
461	ERROR_SMG_INVALID_DATA_LENGTH
478	ERROR_SMG_INVALID_TRACE_OPTION
491	ERROR_SMG_INVALID_PROGRAM_TYPE
492	ERROR_SMG_INVALID_PGM_CONTROL

# DosStartSession – Start Session

493 ERROR\_SMG\_INVALID\_INHERIT\_OPT  
503 ERROR\_SMG\_INVALID\_DEBUG\_PARMS

Any error code returned from DosOpen, DosLoadModule, and DosExecPgm is also possible from DosStartSession.

## Remarks

When a length of 24 bytes is specified, DosStartSession assumes the iconfile, pgmhandle, sessiontype, pgmcontrol, initxpos, initypos, initxsize, and initysize parameters to be 0. A value of 0 allows the Shell to establish the program title, icon definition, session type, program control, window size, and window position for the specified program.

**Foreground/Background Considerations:** If fgbg = 0 is specified, and if neither the calling program nor any of its descendant sessions is executing in the foreground, the new session is started in the background. The ERROR\_SMG\_START\_IN\_BACKGROUND error code is also returned in this case. The foreground session for windowed applications is the session of the application that owns the window focus.

**Parent/Child Relationships:** When related = 1 is specified, DosStartSession establishes a parent session/child session relationship. A parent process/child process relationship is not established. The parent process is the Shell process, just as if the operator had started the program from the Shell menu. The ProcID returned by DosStartSession may not be used by any OS/2 calls (for example, DosSetPrty) that require a parent process/child process relationship. Once a process has issued a DosStartSession, specifying related = 1, no other process within that session may issue related DosStartSession calls until all the dependent sessions have terminated.

**Debugger Considerations:** Debuggers may want to debug all processes associated with an application, no matter how the process was started (DosExecPgm or DosStartSession). A special trace option, traceopt = 2, has been provided for this purpose. When traceopt = 2 is specified, the debugger must also supply the name of an existing queue as the termination queue name and related = 1 on the DosStartSession call.

The Session Manager notifies the debugger whenever a new session is created through DosStartSession from the initial session started with traceopt = 2 or from any descending session. The queue is posted regardless of how the new session is started: related, independent, with or without inheritance, and is executed for tracing. It is the responsibility of the debugger to resume the new process' execution through DosPtrace.

The debugger must issue DosReadQueue to receive notification when a child session is created. The word containing the request parameter returned by DosReadQueue is one. The data element structure returned has the following format:

Size	Definition
WORD	Session ID
WORD	Process ID
WORD	Parent Session ID
WORD	Parent Process ID

DosReadQueue, with the NoWait parameter set to zero, should be issued by the debugger. This is the only process that has addressability to the notification data element. After reading and processing the data element, the debugger must free the segment containing the data element using DosFreeSeg.

The debugger may use DosSelectSession to switch itself or any descendant session into the foreground whenever the current foreground session is a descendant of the debugger.

Some debuggers may enhance usability by using more than one display. Therefore, when a debugger registers with the Session Manager by using a traceopt of 2, the debugger is allowed to update the physical video buffer in the range of B000-B7FF, as long as the foreground session is a descendant of the debugger. The debugger is not allowed to update the physical video buffer when a session is switched into the foreground that is not a descendant of the debugger or when a popup occurs.



# DosStartSession —

## Start Session

**Program Name/Program Input Considerations:** The program identified by pgmname is executed directly with no intermediate secondary command (CMD.EXE) process. Alternatively, the program can be executed indirectly through a secondary command (CMD.EXE) process by specifying CMD.EXE for pgmname and by specifying either /C or /K followed by the drive, path, and file name of the application to be loaded for pgminputs. If the /C parameter is inserted at the beginning of the pgminputs string, the session terminates when the application program terminates. If the /K parameter is inserted at the beginning of the pgminputs string, the operator sees the OS/2 command line prompt displayed when the application terminates. The operator can then either enter the name of another program or command to execute or enter the OS/2 EXIT command to terminate the session.

When the pgmname is accessed with a far address of 0 or the ASCIIZ string is null, the program specified as a parameter to the Shell in the PROTSHELL statement in the CONFIG.SYS file is executed and passed the specified pgminputs. This is the OS/2 mode command processor (CMD.EXE) by default.

When pgmname is equal to the command processor named on the PROTSHELL statement in CONFIG.SYS, or when the pgmname = NULL and length = 24 or 30 bytes, either the command processor named in CONFIG.SYS or the default CMD.EXE is started within the same session as the caller of DosStartSession.

**Program Handle Considerations:** If a process issues a DosStartSession specifying only the program handle, it must change to the working directory before it issues the DosStartSession, and start the new process inherited. If a process is started with inheritopt = 0, that process must change to the correct directory.

**Parent/Child Termination Considerations:** A parent session has only one termination queue. The parent creates the termination queue before it issues its first DosStartSession call specifying the name of the queue. An application can use the termination queue for another purpose by passing a unique request parameter through the DosWriteQueue/DosReadQueue interface. Request parameter values 0 through 99 are reserved for OS/2. Request parameter values greater than 99 are available for application use.

If a parent session specifies the termq parameter on more than one DosStartSession call, the parameter is ignored on subsequent calls. Once a parent establishes a termination queue, the queue is posted when any child session terminates. The queue is posted regardless of who terminates the child session (for example, child, parent, or operator) and whether the termination is normal or abnormal. The termq data element structure has the following format:

Size	Description
WORD	Session ID of child
WORD	Result code.

When a child session terminates, the result code returned in the termq data element is the result code of the program specified by pgmname assuming either:

1. The program is executed directly with no intermediate secondary command (CMD.EXE) process, or
2. The program is executed indirectly through a secondary command (CMD.EXE) process and the /C parameter is specified.

If the program is executed indirectly through a secondary command (CMD.EXE) process and the /K parameter is specified, the result code of the processed command is returned.

When a child session running in the foreground terminates, the parent session becomes the foreground session. When a parent session terminates, any child sessions it created with DosStartSession, specifying related = 1, are terminated. When an independent session, created specifying related = 0, running in the foreground terminates, the Shell chooses the next foreground session.

**Descendant Considerations:** A session started through DosStartSession may in turn issue DosStartSession. These rules apply:

- The SessID specified on DosSelectSession, DosSetSession, and DosStopSession may be only the SessID of an immediate child session, not a grandchild session, and so forth.

## DosStartSession – Start Session

- Suppose a bond is established between session A and its immediate child session B, and another bond is established between session B and its immediate child session C. When the operator selects session A, session C is brought to the foreground. See DosSetSession for a description of what establishing a bond means.
- When a session terminates, all of its descendants (children, grandchildren, and so forth) are terminated.

### C Language

```
typedef struct _STARTDATA {    /* stdata */

    USHORT cb;                /* length of data structure in bytes */
    USHORT Related;           /* 0=independent session, 1=child session */
    USHORT FgBg;              /* 0=start in foreground, 1=start in background */
    USHORT TraceOpt;          /* 0=no trace, 1=trace */
    PSZ PgmTitle;             /* address of program title */
    PSZ PgmName;              /* address of program name */
    PBYTE PgmInputs;          /* input arguments */
    PBYTE TermQ;              /* address of program queue name */
    PBYTE Environment;        /* address of environment string */
    USHORT InheritOpt;        /* inherit option (shell of program) */
    USHORT SessionType;       /* session type (standard, windowed, ...) */
    PSZ IconFile;             /* address of icon definition */
    ULONG PgmHandle;          /* program handle */
    USHORT PgmControl;        /* initial state of windowed application */
    USHORT InitXPos;          /* x coordinate of initial session window */
    USHORT InitYPos;          /* y coordinate of initial session window */
    USHORT InitXSize;         /* initial size of x */
    USHORT InitYSize;         /* initial size of y */

} STARTDATA;

#define INCL_DOSSESMGR

USHORT rc = DosStartSession(StartData, SessID, PID);

PSTARTDATA    StartData;    /* Start session data */
PUSHORT        SessID;       /* Session ID (returned) */
PUSHORT        PID;         /* Process ID (returned) */

USHORT        rc;           /* return code */
```

# DosStartSession — Start Session

## Assembler Language

STARTDATA struc

```
stdata_Length      dw ? ;length of data structure in bytes
stdata_Related      dw ? ;0=independent session, 1=child session
stdata_FgBg         dw ? ;0=start in foreground, 1=start in background
stdata_TraceOpt      dw ? ;0=no trace, 1=trace
stdata_PgmTitle      dd ? ;address of program title
stdata_PgmName       dd ? ;address of program name
stdata_PgmInputs     dd ? ;input arguments
stdata_TermQ         dd ? ;address of program queue name
stdata_Environment  dd ? ;address of environment string
stdata_InheritOpt    dw ? ;inherit option (shell of program)
stdata_SessionType   dw ? ;session type (standard, windowed, ...)
stdata_IconFile      dd ? ;address of icon definition
stdata_PgmHandle     dd ? ;program handle
stdata_PgmControl    dw ? ;initial state of windowed application
stdata_InitXPos      dw ? ;x coordinate of initial session window
stdata_InitYPos      dw ? ;y coordinate of initial session window
stdata_InitXSize     dw ? ;initial size of x
stdata_InitYSize     dw ? ;initial size of y
```

STARTDATA ends

```
EXTRN DosStartSession:FAR
INCL_DOSSESMGR      EQU 1
```

```
PUSH@ OTHER   StartData      ;Start session data
PUSH@ WORD     SessID         ;Session ID (returned)
PUSH@ WORD     PID           ;Process ID (returned)
CALL  DosStartSession
```

Returns WORD

# DosStopSession – Stop Session

This call terminates a session previously started with DosStartSession.

**DosStopSession (TargetOption, SessID, Reserved)**

## Parameters

**TargetOption (USHORT)** – input

Specifies whether the session specified by SessID or all sessions should be terminated.

Value	Definition
0	Terminate session specified.
1	Terminate all child sessions and descendant sessions.

**SessID (USHORT)** – input

Session ID session to be terminated. The value specified must equal the SessID returned on a previous DosStartSession call. SessID is ignored if TargetOption = 1.

**Reserved (ULONG)** – input

A doubleword of 0's.

**rc (USHORT)** – return

Return code descriptions are:

0	NO_ERROR
369	ERROR_SMG_INVALID_SESSION_ID
418	ERROR_SMG_INVALID_CALL
452	ERROR_SMG_SESSION_NOT_PARENT
458	ERROR_SMG_INVALID_STOP_OPTION
459	ERROR_SMG_BAD_RESERVE

## Remarks

DosStopSession may be issued by a parent session only for a child session. The parent session cannot issue this call with itself as the target or a descendant of a child session as the target. However, if the child session specified with DosStopSession does have descendants, these sessions are also terminated.

DosStopSession may terminate only child sessions originally started by the caller with DosStartSession specifying Related = 1. Sessions started as independent sessions cannot be stopped by this call.

A parent session may be running in either the foreground or background when DosStopSession is issued. If a child session is running in the foreground at the time it is stopped, the parent session becomes the foreground session. DosStopSession breaks any bond that was established between the parent session and child session by a DosSetSession request.

The process running in the session specified on the DosStopSession call may refuse to terminate. DosStopSession returns a normal return code in this case. The only way to ensure that a target session has terminated is to wait for notification of its demise by a termination queue created with a DosStartSession request.

## C Language

```
#define INCL_DOSSESMGR
```

```
USHORT rc = DosStopSession(TargetOption, SessID, Reserved);

USHORT TargetOption; /* Target option */
USHORT SessID;       /* Session ID */
ULONG  0;             /* Reserved (must be zero) */

USHORT rc;            /* return code */
```

# DosStopSession — Stop Session

## Assembler Language

```
EXTRN DosStopSession:FAR
INCL_DOSSESMGR EQU 1

PUSH WORD TargetOption ;Target option
PUSH WORD SessID ;Session ID
PUSH DWORD 0 ;Reserved (must be zero)
CALL DosStopSession

Returns WORD
```

This call suballocates portions of a segment allocated by DosAllocSeg or DosAllocShrSeg, and initialized by DosSubSet.

**DosSubAlloc** (*SegSelector*, *BlockOffset*, *Size*)

## Parameters

**SegSelector** (*SEL*) — input

Data segment selector that allocates the memory.

**BlockOffset** (*PUSHORT*) — output

Address of the allocated block offset.

**Size** (*USHORT*) — input

Memory block size requested in bytes.

**rc** (*USHORT*) — return

Return code descriptions are:

<b>0</b>	<b>NO_ERROR</b>
<b>311</b>	<b>ERROR_DOSSUB_NOMEM</b>
<b>313</b>	<b>ERROR_DOSSUB_BADSIZE</b>

## Remarks

Before a segment allocated by DosAllocSeg or DosAllocShrSeg can be suballocated, it must first be initialized for suballocation by a call to DosSubSet.

Allocation size must be a multiple of four bytes. Otherwise, it is rounded up to a multiple of four bytes. The maximum value for the size parameter is the size that was set with DosSubSet minus 8. Note that no paragraph (16-byte) alignment is required; all requests are serviced on a byte alignment basis.

A suballocated block of memory in a suballocated segment is freed by a call to DosSubFree.

## C Language

```
#define INCL_DOSMEMMGR

USHORT rc = DosSubAlloc(SegSelector, BlockOffset, Size);

SEL      SegSelector; /* Segment selector */
PUSHORT  BlockOffset; /* Block Offset (returned) */
USHORT   Size;        /* Size of requested block */

USHORT   rc;          /* return code */
```

## Assembler Language

```
EXTRN DosSubAlloc:FAR
INCL_DOSMEMMGR EQU 1

PUSH WORD SegSelector ;Segment selector
PUSH@ WORD BlockOffset ;Block Offset (returned)
PUSH WORD Size ;Size of requested block
CALL DosSubAlloc
```

Returns WORD

# DosSubFree —

## Free Memory Suballocated Within Segment

FAPI

This call frees memory previously allocated by DosSubAlloc.

**DosSubFree (SegSelector, BlockOffset, Size)**

### Parameters

**SegSelector (SEL)** — input  
Data segment selector.

**BlockOffset (USHORT)** — input  
Memory block offset. The value specified must equal the BlockOffset returned on a previous DosSubAlloc call.

**Size (USHORT)** — input  
Size, in bytes, of the block to be freed.

**rc (USHORT)** — return  
Return code descriptions are:

<b>0</b>	<b>NO_ERROR</b>
<b>312</b>	<b>ERROR_DOSSUB_OVERLAP</b>
<b>313</b>	<b>ERROR_DOSSUB_BADSIZE</b>

### Remarks

DosSubFree specifies the offset of a block of memory previously suballocated by a DosSubAlloc request. If the block specified overlaps memory in the segment that is not suballocated, an error is returned. Like DosSubAlloc, the size parameter must be a multiple of four bytes; otherwise, it is rounded up to a multiple of four bytes.

The allocated segment is freed by calling DosFreeSeg.

### C Language

```
#define INCL_DOSMEMMGR

USHORT rc = DosSubFree(SegSelector, BlockOffset, Size);

SEL      SegSelector; /* Segment selector */
USHORT   BlockOffset; /* Offset of memory block to free */
USHORT   Size;        /* Size of block in bytes */

USHORT   rc;          /* return code */
```

### Assembler Language

```
EXTRN DosSubFree:FAR
INCL_DOSMEMMGR EQU 1

PUSH WORD SegSelector ;Segment selector
PUSH WORD BlockOffset ;Offset of memory block to free
PUSH WORD Size ;Size of block in bytes
CALL DosSubFree

Returns WORD
```

This call is used to initialize a segment or to reset a reallocated segment for suballocation.

**DosSubSet (SegSelector, Flags, Size)**

## Parameters

**SegSelector (SEL)** – input  
Target data segment selector.

**Flags (USHORT)** – input  
0 = Increasing the size of a segment already initialized.  
1 = Initializing a segment.

**Size (USHORT)** – input  
Segment size in bytes.

**rc (USHORT)** – return  
Return code descriptions are:

0	NO_ERROR
310	ERROR_DOSSUB_SHRINK
313	ERROR_DOSSUB_BADSIZE
314	ERROR_DOSSUB_BADFLAG

## Remarks

To initialize a segment for suballocation, issue DosSubSet before issuing DosSubAlloc and set Flags = 1. The segment must have been allocated with DosAllocSeg or DosAllocShrSeg.

If a segment allocated by a DosAllocSeg call has already been set for suballocation, and a call to DosSubAlloc returns Error\_DOSSUB\_NOMEM, the segment's size can be increased by a call to DosReallocSeg. After reallocation, the segment must be reset by a DosSubSet. Failure to reset the segment after changing its size can yield unpredictable results.

The size parameter should be a multiple of four bytes, or it is rounded up to a multiple of four. Note in DosSubSet, a size parameter of 0 indicates the segment is 64KB, while in DosSubAlloc and DosSubFree, a size parameter of 0 is an error. Other than this special case of 0 meaning 64KB, the minimum size that can be set is 12 bytes.

## C Language

```
#define INCL_DOSMEMMGR

USHORT rc = DosSubSet(SegSelector, Flags, Size);

SEL      SegSelector; /* Segment selector */
USHORT   Flags;       /* Parameter flags */
USHORT   Size;        /* Size of a block */

USHORT   rc;          /* return code */
```



# DosSubSet — Initialize or Set Allocated Memory

FAP1

## Assembler Language

```
EXTRN DosSubSet:FAR
INCL_DOSMEMMGR EQU 1

PUSH WORD SegSelector ;Segment selector
PUSH WORD Flags ;Parameter flags
PUSH WORD Size ;Size of a segment
CALL DosSubSet

Returns WORD
```

# DosSuspendThread – Suspend Thread Execution

This call temporarily suspends execution of another thread within the current process until a `DosResumeThread` call is issued.

**DosSuspendThread (ThreadID)**

## Parameters

**ThreadID (TID)** – input  
Thread ID to be suspended.

**rc (USHORT)** – return  
Return code descriptions are:

0	NO_ERROR
309	ERROR_INVALID_THREADID

## Remarks

A thread's execution is suspended when another thread in its process issues `DosSuspendThread`, specifying the ID of the target thread. The thread may not be suspended immediately because it may have locked some system resources that have to be freed first. However, the thread is not allowed to execute further application program instructions until a corresponding `DosResumeThread` is issued.

`DosSuspendThread` permits the suspension of only one other thread within the current process. If a thread needs to disable all thread switching within its process so the calling thread can execute time-critical code, it uses `DosEnterCritSec` and `DosExitCritSec` calls.

## C Language

```
#define INCL_DOSPROCESS

USHORT rc = DosSuspendThread(ThreadID);

TID          ThreadID;    /* Thread ID */

USHORT       rc;          /* return code */
```

## Assembler Language

```
EXTRN DosSuspendThread:FAR
INCL_DOSPROCESS EQU 1

PUSH WORD ThreadID ;Thread ID
CALL DosSuspendThread

Returns WORD
```

## Example

The following example shows how to suspend and resume execution of a thread within a process. The main thread creates `Thread2` and allows it to begin executing. `Thread2` iterates through a loop that prints a line and then sleeps, relinquishing its time slice to the main thread. After one iteration by `Thread2`, the main thread suspends `Thread2` and then resumes it. Subsequently, `Thread2` completes the remaining three iterations.

## DosSuspendThread – Suspend Thread Execution

```
#define INCL_DOSPROCESS

#include <os2.h>

#define SEGSIZE      4000  /* Number of bytes requested in segment */
#define ALLOCFLAGS    0    /* Segment allocation flags - no sharing */
#define SLEEPSHORT    5L   /* Sleep interval - 5 milliseconds */
#define SLEEPLONG     75L  /* Sleep interval - 75 milliseconds */
#define RETURN_CODE   0    /* Return code for DosExit() */

VOID APIENTRY Thread2()
{
    USHORT    i;

    /* Loop with four iterations */
    for(i=1; i<5; i++)
    {
        printf("In Thread2, i is now %d\n", i);

        /* Sleep to relinquish time slice to main thread */
        DosSleep(SLEEPSHORT);      /* Sleep interval */
    }
    DosExit(EXIT_THREAD,          /* Action code - end a thread */
            RETURN_CODE);         /* Return code */
}

main()
{
    TID        ThreadID;          /* Thread identification */
    SEL        ThreadStackSel;    /* Segment selector for thread stack */
    PBYTE      StackEnd;          /* Ptr. to end of thread stack */
    USHORT     rc;

    /** Allocate segment for thread stack; make pointer to end of stack. **/
    /** We must allocate a segment in order to preserve segment protection **/
    /** for the thread. **/

    rc = DosAllocSeg(SEGSIZE,      /* Number of bytes requested */
                    &ThreadStackSel, /* Segment selector (returned) */
                    ALLOCFLAGS);    /* Allocation flags - no sharing */
    StackEnd = MAKEP(ThreadStackSel, SEGSIZE-1);

    /** Start Thread2 **/
    if(!rc=DosCreateThread((PFNTHREAD) Thread2, /* Thread address */
                          &ThreadID,           /* Thread ID (returned) */
                          StackEnd))           /* End of thread stack */
        printf("Thread2 created.\n");

    /* Sleep to relinquish time slice to Thread2 */
    if(!DosSleep(SLEEPSHORT)) /* Sleep interval */
        printf("Slept a little to let Thread2 execute.\n");

    /***** Suspend Thread2, do some work, then resume Thread2 *****/
    if(!rc=DosSuspendThread(ThreadID)) /* Thread ID */
        printf("Thread2 SUSPENDED.\n");
    printf("Perform work that will not be interrupted by Thread2.\n");
    if(!rc=DosResumeThread(ThreadID)) /* Thread ID */
        printf("Thread2 RESUMED.\n");
    printf("Now we may be interrupted by Thread2.\n");

    /* Sleep to allow Thread2 to complete */
    DosSleep(SLEEPLONG);      /* Sleep interval */
}
```

# DosTimerAsync – Start Asynchronous Time Delay

This call starts a timer that runs asynchronously to the thread issuing the request and clears a system semaphore when the specified interval expires.

**DosTimerAsync** (TimeInterval, SemHandle, Handle)

## Parameters

**TimeInterval** (*ULONG*) – input

Number of milliseconds (rounded up to the next clock tick) that passes before the semaphore is cleared.

**SemHandle** (*HSEM*) – input

Handle of the system semaphore used to communicate the time out to the calling thread. This semaphore should be set by DosSemSet before DosTimerAsync is called.

**Handle** (*PHTIMER*) – output

Address of the timer handle. This handle may be passed to DosTimerStop to stop this timer before its time interval expires.

**rc** (*USHORT*) – return

Return code descriptions are:

0	NO_ERROR
323	ERROR_TS_SEMHANDLE
324	ERROR_TS_NOTIMER

## Remarks

DosTimerAsync is used to wait for a single asynchronous time. The thread waits for the time out by issuing a DosSemWait.

This function is the asynchronous analog of DosSleep. This function allows a thread to start a timer while it is performing another task. This timer can be canceled by calling the DosTimerStop function with the timer handle returned by DosTimerAsync.

If another time out is needed, the semaphore is set and DosTimerAsync is reissued. To ensure reliable detection of the timer expiration, the system semaphore should be set prior to calling DosTimerAsync.

To set a periodic interval timer, see DosTimerStart.

## C Language

```
#define INCL_DOSDATETIME
```

```
USHORT rc = DosTimerAsync(TimeInterval, SemHandle, Handle);
```

```
ULONG      TimeInterval; /* Interval size (in milliseconds) */
HSEM       SemHandle;    /* System semaphore handle */
PHTIMER    Handle;       /* Timer handle (returned) */

USHORT     rc;           /* return code */
```

# DosTimerAsync – Start Asynchronous Time Delay

## Assembler Language

```
EXTRN DosTimerAsync:FAR
INCL_DOSDATETIME EQU 1

PUSH  DWORD   TimeInterval ;Interval size (in milliseconds)
PUSH  DWORD   SemHandle    ;System semaphore handle
PUSH@  WORD    Handle       ;Timer handle (returned)
CALL  DosTimerAsync
```

Returns WORD

## Example

The following example sets an asynchronous one-shot timer for one second. It then sets an asynchronous recurring timer with a one-second interval, reporting each time an interval elapses. Finally, it stops the recurring timer.

```
#define INCL_DOSDATETIME
#define INCL_DOSSEMAPHORES

#include <os2.h>

#define SEM_NAME      "\\SEM\\timer.sem" /* Semaphore name */
#define TIME_INTERVAL 1000L             /* Timer interval (in milliseconds) */

main()
{
    HSEM    SemHandle;
    HTIMER  TimerHandle;
    USHORT  i;
    USHORT  rc;

    /* Create system semaphore to be used by timers */
    DosCreateSem(CSEM_PUBLIC, /* Ownership - nonexclusive */
                &SemHandle, /* Semaphore handle (returned) */
                SEM_NAME); /* Semaphore name */

    /* Set the semaphore, then start a one-shot timer */
    if(!DosSemSet(SemHandle)) /* Semaphore handle */
        printf("Semaphore set.\n");
    if(!rc=DosTimerAsync(TIME_INTERVAL, /* Timer interval */
                        SemHandle, /* Semaphore handle */
                        &TimerHandle)) /* Timer handle (returned) */
        printf("One shot timer for %f seconds started.\n", TIME_INTERVAL/1000.0);

    /* Report when timer expires (other work may be done here) */
    if(!DosSemWait(SemHandle, /* Semaphore handle */
                  SEM_INDEFINITE_WAIT)) /* Timeout period - indefinite */
        printf("Time interval has elapsed.\n");

    /* Start a recurring timer */
    if(!rc=DosTimerStart(TIME_INTERVAL, /* Timer interval */
                        SemHandle, /* Semaphore handle */
                        &TimerHandle)) /* Timer handle (returned) */
        printf("Recurring timer with %f second interval started.\n",
              TIME_INTERVAL/1000.0);

    /* */
    for(i=1; i<4; i++)
        if(!DosSemSetWait(SemHandle, /* Semaphore handle */
                          SEM_INDEFINITE_WAIT)) /* Timeout period - indefinite */
            printf("Recurring timer cleared semaphore %d times.\n", i);
    if(!rc=DosTimerStop(TimerHandle)) /* Timer handle */
        printf("Recurring timer has been stopped.\n");
}
```

# DosTimerStart – Start Periodic Interval Timer

This call starts a periodic interval timer that runs asynchronously to the thread issuing the request. The semaphore is continually cleared at the specified time interval until the timer is turned off by DosTimerStop.

**DosTimerStart (TimeInterval, SemHandle, Handle)**

## Parameters

**TimeInterval (ULONG)** – input

Number of milliseconds (rounded up to the next clock tick) that passes before the semaphore is cleared.

**SemHandle (HSEM)** – input

Handle of the system semaphore used to communicate the time out to the calling thread. This semaphore should be set by DosSemSet before the next clear by the timer.

**Handle (PHTIMER)** – output

Address of the timer handle. This handle may be passed to DosTimerStop to stop the periodic timer.

**rc (USHORT)** – return

Return code descriptions are:

0	NO_ERROR
323	ERROR_TS_SEMHANDLE
324	ERROR_TS_NOTIMER

## Remarks

DosTimerStart allows a timer to start and run asynchronously to a thread. A timer interval is canceled by using the timer handle in the DosTimerStop call. This prevents the semaphore indicated in the DosTimerStart call from being sent notifications.

The application detects the expirations of the timer when the semaphore is set prior to the next expiration of the timer. When an application waits for this semaphore to clear, more than one clearing of the timer may occur before the application resumes execution. If it is necessary to determine the actual elapsed time, the Global Information Segment milliseconds field can be saved by a DosGetInfoSeg request before calling DosTimerStart. This saved value is compared to the current value when the process resumes.

## C Language

```
#define INCL_DOSDATETIME
```

```
USHORT rc = DosTimerStart(TimeInterval, SemHandle, Handle);
```

```
ULONG      TimeInterval; /* Interval size (in milliseconds) */
HSEM       SemHandle;    /* System semaphore handle */
PHTIMER    Handle;       /* Timer handle (returned) */

USHORT     rc;           /* return code */
```

## Assembler Language

```
EXTRN DosTimerStart:FAR
INCL_DOSDATETIME EQU 1

PUSH DWORD TimeInterval ;Interval size (in milliseconds)
PUSH DWORD SemHandle     ;System semaphore handle
PUSH@ WORD Handle        ;Timer handle (returned)
CALL DosTimerStart
```

Returns WORD

# DosTimerStart —

## Start Periodic Interval Timer

### Example

The following example sets an asynchronous one-shot timer for one second. It then sets an asynchronous recurring timer with a one-second interval, reporting each time an interval elapses. Finally, it stops the recurring timer.

```
#define INCL_DOSDATETIME
#define INCL_DOSSEMAPHORES

#include <os2.h>

#define SEM_NAME      "\\SEM\\timer.sem" /* Semaphore name */
#define TIME_INTERVAL 1000L           /* Timer interval (in milliseconds) */

main()
{
    HSEM    SemHandle;
    HTIMER  TimerHandle;
    USHORT  i;
    USHORT  rc;

    /* Create system semaphore to be used by timers */
    DosCreateSem(CSEM_PUBLIC, /* Ownership - nonexclusive */
                &SemHandle, /* Semaphore handle (returned) */
                SEM_NAME); /* Semaphore name */

    /* Set the semaphore, then start a one-shot timer */
    if(!DosSemSet(SemHandle) /* Semaphore handle */
        printf("Semaphore set.\n");
    if(!rc=DosTimerAsync(TIME_INTERVAL, /* Timer interval */
                        SemHandle, /* Semaphore handle */
                        &TimerHandle)) /* Timer handle (returned) */
        printf("One shot timer for %f seconds started.\n", TIME_INTERVAL/1000.0);

    /* Report when timer expires (other work may be done here) */
    if(!DosSemWait(SemHandle, /* Semaphore handle */
                  SEM_INDEFINITE_WAIT) /* Timeout period - indefinite */
        printf("Time interval has elapsed.\n");

    /* Start a recurring timer */
    if(!rc=DosTimerStart(TIME_INTERVAL, /* Timer interval */
                        SemHandle, /* Semaphore handle */
                        &TimerHandle)) /* Timer handle (returned) */
        printf("Recurring timer with %f second interval started.\n",
              TIME_INTERVAL/1000.0);

    /* */
    for(i=1; i<4; i++)
        if(!DosSemSetWait(SemHandle, /* Semaphore handle */
                          SEM_INDEFINITE_WAIT) /* Timeout period - indefinite */
            printf("Recurring timer cleared semaphore %d times.\n", i);
    if(!rc=DosTimerStop(TimerHandle)) /* Timer handle */
        printf("Recurring timer has been stopped.\n");
}
```

## DosTimerStop – Stop Interval Timer

---

This call stops a periodic interval timer started by DosTimerStart, or an asynchronous timer started by DosTimerAsync.

<b>DosTimerStop (Handle)</b>
------------------------------

### Parameters

**Handle (HTIMER)** – input  
Handle of the timer to be stopped.

**rc (USHORT)** – return  
Return code descriptions are:

0	NO_ERROR
326	ERROR_TS_HANDLE

### Remarks

DosTimerStop is used to stop an asynchronous one-shot timer started with DosTimerAsync or an asynchronous periodic interval timer started with DosTimerStart.

No assumptions can be made about the state of the semaphore specified with DosTimerStart or DosTimerAsync. The application should put the semaphores into a known state.

### C Language

```
#define INCL_DOSDATETIME

USHORT rc = DosTimerStop(Handle);

HTIMER      Handle;      /* Handle of the timer */

USHORT      rc;          /* return code */
```

### Assembler Language

```
EXTRN DosTimerStop:FAR
INCL_DOSDATETIME EQU 1

PUSH WORD Handle ;Handle of the timer
CALL DosTimerStop
```

Returns WORD

### Example

The following example sets an asynchronous one-shot timer for one second. It then sets an asynchronous recurring timer with a one-second interval, reporting each time an interval elapses. Finally, it stops the recurring timer.



## DosTimerStop — Stop Interval Timer

```
#define INCL_DOSDATETIME
#define INCL_DOSSEMAPHORES

#include <os2.h>

#define SEM_NAME      "\\SEM\\timer.sem" /* Semaphore name */
#define TIME_INTERVAL 1000L             /* Timer interval (in milliseconds) */

main()
{
    HSEM    SemHandle;
    HTIMER  TimerHandle;
    USHORT  i;
    USHORT  rc;

    /* Create system semaphore to be used by timers */
    DosCreateSem(CSEM_PUBLIC, /* Ownership - nonexclusive */
                &SemHandle, /* Semaphore handle (returned) */
                SEM_NAME); /* Semaphore name */

    /* Set the semaphore, then start a one-shot timer */
    if(!DosSemSet(SemHandle)) /* Semaphore handle */
        printf("Semaphore set.\n");
    if(!rc=DosTimerAsync(TIME_INTERVAL, /* Timer interval */
                        SemHandle, /* Semaphore handle */
                        &TimerHandle)) /* Timer handle (returned) */
        printf("One shot timer for %f seconds started.\n", TIME_INTERVAL/1000.0);

    /* Report when timer expires (other work may be done here) */
    if(!DosSemWait(SemHandle, /* Semaphore handle */
                  SEM_INDEFINITE_WAIT)) /* Timeout period - indefinite */
        printf("Time interval has elapsed.\n");

    /* Start a recurring timer */
    if(!rc=DosTimerStart(TIME_INTERVAL, /* Timer interval */
                        SemHandle, /* Semaphore handle */
                        &TimerHandle)) /* Timer handle (returned) */
        printf("Recurring timer with %f second interval started.\n",
              TIME_INTERVAL/1000.0);

    /* */
    for(i=1; i<4; i++)
        if(!DosSemSetWait(SemHandle, /* Semaphore handle */
                          SEM_INDEFINITE_WAIT)) /* Timeout period - indefinite */
            printf("Recurring timer cleared semaphore %d times.\n", i);
    if(!rc=DosTimerStop(TimerHandle)) /* Timer handle */
        printf("Recurring timer has been stopped.\n");
}
```

# DosTransactNmPipe – Perform Transaction

This call performs a write followed by a read on a duplex message pipe.

**DosTransactNmPipe** (*Handle*, *InBuffer*, *InBufferLen*, *OutBuffer*, *OutBufferLen*, *BytesOut*)

## Parameters

**Handle** (*HPIPE*) – input

Named pipe handle returned by `DosMakeNmPipe` or `DosOpen`.

**InBuffer** (*PBYTE*) – input

Address of the buffer to write to the pipe.

**InBufferLen** (*USHORT*) – input

Number of bytes to be written.

**OutBuffer** (*PBYTE*) – output

Address of the buffer for returned data.

**OutBufferLen** (*USHORT*) – input

Maximum size, number of bytes, of returned data.

**BytesOut** (*PUSHORT*) – output

Address of the number of bytes read.

**rc** (*USHORT*) – return

Return code descriptions are:

<b>0</b>	<b>NO_ERROR</b>
<b>11</b>	<b>ERROR_BAD_FORMAT</b>
<b>230</b>	<b>ERROR_BAD_PIPE</b>
<b>231</b>	<b>ERROR_PIPE_BUSY</b>
<b>233</b>	<b>ERROR_PIPE_NOT_CONNECTED</b>
<b>234</b>	<b>ERROR_MORE_DATA</b>

## Remarks

`DosTransactNmPipe` is intended for use only on duplex message pipes. If this call is issued and the pipe is not a duplex message pipe, `ERROR_BAD_FORMAT` is returned.

This call enables you to implement transaction-oriented dialogs. `DosTransactNmPipe` writes the entire `InBuffer` to the pipe and then reads a response from the pipe into the `OutBuffer`. The current state of blocking/non-blocking has no effect. `DosTransactNmPipe` does not return until a message has been read into the `OutBuffer`. If the `OutBuffer` is too small to contain the response message, `ERROR_MORE_DATA` is returned, as described for `DosRead`.

## C Language

```
#define INCL_DOSNMPICES
```

```
USHORT rc = DosTransactNmPipe(Handle, InBuffer, InBufferLen, OutBuffer,  
                               OutBufferLen, BytesOut);
```

```
HPIPE      Handle;      /* Pipe handle */  
PBYTE      InBuffer;    /* Write buffer */  
USHORT     InBufferLen; /* Write buffer length */  
PBYTE      OutBuffer;   /* Read buffer (returned) */  
USHORT     OutBufferLen; /* Read buffer length */  
PUSHORT     BytesOut;   /* Bytes read (returned) */  
  
USHORT     rc;          /* return code */
```

# DosTransactNmPipe — Perform Transaction

## Assembler Language

```
EXTRN DosTransactNmPipe:FAR
INCL_DOSNMPICES EQU 1

PUSH WORD Handle ;Pipe handle
PUSH@ OTHER InBuffer ;Write buffer
PUSH WORD InBufferLen ;Write buffer length
PUSH@ OTHER OutBuffer ;Read buffer (returned)
PUSH WORD OutBufferLen ;Read buffer length
PUSH@ WORD BytesOut ;Bytes read (returned)
CALL DosTransactNmPipe

Returns WORD
```

# DosUnlockSeg – Unlock Segment

---

This call unlocks a discardable segment.

<b>DosUnlockSeg (Selector)</b>
--------------------------------

## Parameters

**Selector (SEL)** – input  
Segment selector to be unlocked.

**rc (USHORT)** – return  
Return code descriptions are:

<b>0</b>	<b>NO_ERROR</b>
<b>5</b>	<b>ERROR_ACCESS_DENIED</b>
<b>158</b>	<b>ERROR_NOT_LOCKED</b>

## Remarks

DosUnlockSeg is called to free memory for possible discard by the system in a low memory situation. The memory being freed is originally allocated by a call to DosAllocSeg or DosAllocHuge with AllocFlags bit 2 set. This memory may have been reallocated by a call to DosReallocSeg or DosReallocHuge after discard by the system.

Allocation and reallocation calls for discardable memory automatically lock the memory for access by the calling process. Thus, to access the segment, the caller does not have to lock the segment with DosLockSeg. Once a discardable segment is unlocked by a DosUnlockSeg request, access to the segment is gained by a successful DosLockSeg request.

DosUnlockSeg may also be used on segments that are non-discardable, in which case it has no effect.

## C Language

```
#define INCL_DOSMEMMGR

USHORT rc = DosUnlockSeg(Selector);

SEL      Selector;      /* Selector to unlock */

USHORT    rc;           /* return code */
```

## Assembler Language

```
EXTRN DosUnlockSeg:FAR
INCL_DOSMEMMGR EQU 1

PUSH WORD Selector      ;Selector to unlock
CALL DosUnlockSeg

Returns WORD
```

# DosWaitNmPipe — Wait Named Pipe Instance

---

This call waits for the availability of a named pipe instance.

**DosWaitNmPipe (FileName, TimeOut)**

## Parameters

**FileName (PSZ)** — input

Address of the ASCIIZ name of the pipe to be opened. Pipes are name \PIPE\FileName.

**TimeOut (ULONG)** — input

Maximum time in milliseconds to wait for the named pipe to become available. When a zero value is used, the DosMakeNmPipe specified default value is used. When a minus one value is used, an indefinite wait is entered.

**rc (USHORT)** — return

Return code descriptions are:

<b>0</b>	<b>NO_ERROR</b>
<b>2</b>	<b>ERROR_FILE_NOT_FOUND</b>
<b>95</b>	<b>ERROR_INTERRUPT</b>
<b>231</b>	<b>ERROR_PIPE_BUSY</b>

## Remarks

This call should be used only when ERROR\_PIPE\_BUSY is returned from a DosOpen call.

DosWaitNmPipe allows an application to wait for a server on a named pipe for which all instances are currently busy. The call waits up to TimeOut milliseconds (or a default time if TimeOut is zero).

When a pipe instance becomes available, it is given to the process whose thread has the highest priority. The priority of a thread may be changed with DosSetPrty.

## C Language

```
#define INCL_DOSNMPICES
```

```
USHORT rc = DosWaitNmPipe(FileName, TimeOut);
```

```
PSZ      FileName;    /* Pipe name */
ULONG    TimeOut;     /* Maximum wait time */
```

```
USHORT    rc;         /* return code */
```

## Assembler Language

```
EXTRN DosWaitNmPipe:FAR
INCL_DOSNMPICES EQU 1
```

```
PUSH@ ASCIIZ FileName    ;Pipe name
PUSH  DWORD  TimeOut      ;Maximum wait time
CALL  DosWaitNmPipe
```

Returns WORD

This call transfers the specified number of bytes from a buffer to the specified file, synchronously with respect to the requesting process's execution.

**DosWrite (FileHandle, BufferArea, BufferLength, BytesWritten)**

## Parameters

**FileHandle (HFILE)** – input

File handle from DosOpen.

**BufferArea (PVOID)** – input

Address of the output buffer.

**BufferLength (USHORT)** – input

Number of bytes to write.

**BytesWritten (PUSHORT)** – output

Address of the number of bytes written.

**rc (USHORT)** – return

Return code descriptions are:

0	NO_ERROR
5	ERROR_ACCESS_DENIED
6	ERROR_INVALID_HANDLE
26	ERROR_NOT_DOS_DISK
33	ERROR_LOCK_VIOLATION
109	ERROR_BROKEN_PIPE

## Remarks

On output, BytesWritten is the number of bytes actually written. If BytesWritten is different from BufferLength, this usually indicates insufficient disk space.

A BufferLength value of 0 is not considered an error. No data transfer occurs. There is no effect on the file or the file pointer.

Buffers that are multiples of the hardware's base physical unit for data written to the file on these base boundaries, are written directly to the device. (The base physical unit is defined as the smallest block that can be physically written to the device.) Other buffer sizes force some I/O to go through an internal system buffer and greatly reduce the efficiency of I/O operation.

The file pointer is moved by read and write operations. It can be moved to a desired position by calling DosChgFilePtr.

If the file is read-only, the write to the file is not performed.

## Family API Considerations

Some options operate differently in the DOS mode than in OS/2 mode. Therefore, the following restriction applies to DosWrite when coding for the DOS mode.

- Only single-byte DosWrite requests can be made to the COM device, because the COM device driver for the DOS environment does not support multiple-byte I/O.

# DosWrite —

## Synchronous Write to File

FAPI

### Named Pipe Considerations

DosWrite is also used to write bytes or messages to a named pipe.

Each write to a **message pipe** writes a message whose size is the length of the write; DosWrite automatically encodes message lengths in the pipe, so applications need not encode this information in the buffer being written.

Writes in blocking mode always write all requested bytes before returning. In non-blocking mode, if the message size is bigger than the buffer size, the write blocks. If the message size is smaller than the pipe but not enough space is left in the pipe, the write returns immediately with a value of zero, indicating no bytes were written.

In the case of a **byte pipe**, if the number of bytes to be written exceeds the space available in the pipe, DosWrite writes as many bytes as it can and returns with the number of bytes actually written.

An attempt to write to a pipe whose other end has been closed returns ERROR\_BROKEN\_PIPE.

### C Language

```
#define INCL_DOSFILEMGR

USHORT rc = DosWrite(FileHandle, BufferArea, BufferLength, BytesWritten);

HFILE      FileHandle;    /* File handle */
PVOID      BufferArea;     /* User buffer */
USHORT     BufferLength;   /* Buffer length */
PUSHORT    BytesWritten;  /* Bytes written (returned) */

USHORT     rc;            /* return code */
```

### Assembler Language

```
EXTRN DosWrite:FAR
INCL_DOSFILEMGR EQU 1

PUSH WORD FileHandle ;File handle
PUSH@ OTHER BufferArea ;User buffer
PUSH WORD BufferLength ;Buffer length
PUSH@ WORD BytesWritten ;Bytes written (returned)
CALL DosWrite

Returns WORD
```

## Example

This example writes to a file.

```
#define INCL_DOSFILEMGR

#define OPEN_FILE 0x01
#define CREATE_FILE 0x10
#define FILE_ARCHIVE 0x20
#define FILE_EXISTS OPEN_FILE
#define FILE_NOEXISTS CREATE_FILE
#define DASD_FLAG 0
#define INHERIT 0x80
#define WRITE_THRU 0
#define FAIL_FLAG 0
#define SHARE_FLAG 0x10
#define ACCESS_FLAG 0x02

#define FILE_NAME "test.dat"
#define FILE_SIZE 800L
#define FILE_ATTRIBUTE FILE_ARCHIVE
#define RESERVED 0L

HFILE FileHandle;
USHORT Wrote;
USHORT Action;
PSZ FileData[100];
USHORT rc;

Action = 2;
strcpy(FileData, "Data...");
if(!DosOpen(FILE_NAME, /* File path name */
             &FileHandle, /* File handle */
             &Action, /* Action taken */
             FILE_SIZE, /* File primary allocation */
             FILE_ATTRIBUTE, /* File attribute */
             FILE_EXISTS | FILE_NOEXISTS, /* Open function type */
             DASD_FLAG | INHERIT | /* Open mode of the file */
             WRITE_THRU | FAIL_FLAG |
             SHARE_FLAG | ACCESS_FLAG,
             RESERVED)) /* Reserved (must be zero) */
    rc = DosWrite(FileHandle, /* File handle */
                  (PVOID) FileData, /* User buffer */
                  sizeof(FileData), /* Buffer length */
                  &Wrote); /* Bytes written */
```



# DosWriteAsync —

## Asynchronous Write to File

---

This call asynchronously transfers the specified number of bytes from a buffer to a file. from a buffer.

**DosWriteAsync** (**FileHandle**, **RamSemaphore**, **ReturnCode**, **BufferArea**, **BufferLength**, **BytesWritten**)

### Parameters

**FileHandle** (*HFILE*) — input

File handle obtained from DosOpen.

**RamSemaphore** (*PULONG*) — input

Address used by the system to signal the caller that the write operation is complete.

**ReturnCode** (*PUSHORT*) — output

Address of the I/O error return code.

**BufferArea** (*PVOID*) — input

Address of the output buffer.

**BufferLength** (*USHORT*) — input

Number of bytes to be written.

**BytesWritten** (*PUSHORT*) — output

Address of the number of bytes written.

**rc** (*USHORT*) — return

Return code descriptions are:

0	NO_ERROR
5	ERROR_ACCESS_DENIED
6	ERROR_INVALID_HANDLE
26	ERROR_NOT_DOS_DISK
33	ERROR_LOCK_VIOLATION
89	ERROR_NO_PROC_SLOTS
109	ERROR_BROKEN_PIPE

### Remarks

A BufferLength value of 0 is not considered an error. No data transfer occurs. There is no effect on the file or the file pointer.

A call to DosWriteAsync may return before the write is complete. To wait for the asynchronous write to complete, RamSemaphore must be set by the application before the DosWriteAsync call is made. The application issues DosSemSet to set the semaphore, calls DosWriteAsync, and then issues DosSemWait to wait for the clearing of the semaphore, which signals the write is complete.

When RamSemaphore is cleared, BytesWritten identifies the number of bytes written. If BytesWritten is different from BufferLength, it usually indicates insufficient disk space.

The program must not modify the contents of BufferArea or look at the values returned in ReturnCode or BytesWritten until after RamSemaphore is cleared.

Buffers that are multiples in size of the hardware's base physical unit for data, written to the file on these base boundaries, are written directly to the device. (The base physical unit is defined as the smallest block that can be physically written to the device.) Other buffer sizes force at least some I/O to go through an internal system buffer (if the file handle state bit indicates that internal buffers may be used) and reduce the efficiency of I/O operation.

## DosWriteAsync – Asynchronous Write to File

The read/write pointer is moved by I/O operations. It can also be moved to a desired position by calling DosChgFilePtr. The value of the pointer is updated by the File Level Request Router before the I/O request is queued to the device driver.

If the file is read-only, the write to the file is not performed.

**Note:** If it is necessary to make a DosWriteAsync request from within a segment that has I/O privilege, DosCallback may be used to invoke a privilege level 3 segment, which makes the actual DosWriteAsync request.

### Named Pipe Considerations

DosWriteAsync is also used to write bytes and messages to named pipes.

Each write to a **message pipe** writes a message whose size is the length of the write; DosWriteAsync automatically encodes message lengths in the pipe, so applications need not encode this information in the buffer being written.

Writes in blocking mode always write all requested bytes before returning. In non-blocking mode, if the message size is bigger than the buffer size, the write blocks. If the message size is smaller than the pipe, but not enough space is left in the pipe, DosWriteAsync returns immediately with a value of zero, indicating no bytes were written.

In the case of a **byte pipe**, if the number of bytes to be written exceeds the space available in the pipe, DosWriteAsync writes as many bytes as it can and returns with the number of bytes actually written.

An attempt to write to a pipe whose other end has been closed returns with ERROR\_BROKEN\_PIPE.

### C Language

```
#define INCL_DOSFILEMGR
```

```
USHORT rc = DosWriteAsync(FileHandle, RamSemaphore, ReturnCode,  
                          BufferArea, BufferLength, BytesWritten);  
  
HFILE      FileHandle;    /* File handle */  
PULONG     RamSemaphore;  /* RAM semaphore */  
PUSHORT    ReturnCode;    /* I/O operation return code (returned) */  
PVOID      BufferArea;    /* User buffer */  
USHORT     BufferLength;   /* Buffer length */  
PUSHORT    BytesWritten;  /* Bytes written (returned) */  
  
USHORT     rc;            /* return code */
```

### Assembler Language

```
EXTRN DosWriteAsync:FAR  
INCL_DOSFILEMGR EQU 1  
  
PUSH WORD FileHandle ;File handle  
PUSH@ DWORD RamSemaphore ;Ram semaphore  
PUSH@ WORD ReturnCode ;I/O operation return code (returned)  
PUSH@ OTHER BufferArea ;User buffer  
PUSH WORD BufferLength ;Buffer length  
PUSH@ WORD BytesWritten ;Bytes written (returned)  
CALL DosWriteAsync  
  
Returns WORD
```

# DosWriteQueue — Write to Queue

---

This call adds an element to a queue.

<b>DosWriteQueue</b> ( <i>QueueHandle</i> , <i>Request</i> , <i>DataLength</i> , <i>DataBuffer</i> , <i>ElemPriority</i> )
--

## Parameters

**QueueHandle** (*HQUEUE*) — input  
Queue handle.

**Request** (*USHORT*) — input  
A value to be passed with the queue element. This word is used for event encoding by the specific application.

**DataLength** (*USHORT*) — input  
Length of the data being sent to the queue.

**DataBuffer** (*PBYTE*) — input  
Address of the data buffer where data, that is to be placed in the queue, is located.

**ElemPriority** (*UCHAR*) — input  
Priority of the element being added to the queue. If the priority is specified as 15, the element is added to the top of the queue (that is, in LIFO order). If the priority is specified as 0, the element is added as the last element in the queue (that is, in FIFO order). Elements with the same priority are in FIFO order. This parameter is valid for priority-type queues only.

**rc** (*USHORT*) — return  
Return code descriptions are:

0	NO_ERROR
334	ERROR_QUE_NO_MEMORY
337	ERROR_QUE_INVALID_HANDLE

## Remarks

DosWriteQueue adds entries to a specified queue.

The Request, DataLength and DataBuffer parameters contain data understood by the thread adding the element to the queue and by the thread that receives the queue element. There is no special meaning to this data; applications may use these parameters for any purpose they wish. OS/2 does not alter this data; it simply copies this data intact. OS/2 does not validate the address of DataBuffer or the DataLength.

If the queue owner has defined a semaphore for use in its notification when elements are added to the queue and if that semaphore is a RAM semaphore, then that semaphore must be in a segment which is shared among both the queue owner's process and this process. If that semaphore handle is for a system semaphore, then that semaphore must be opened by this process before making a DosWriteQueue request to the queue.

If the owning process is terminated, or if the queue is closed before this request is issued, ERROR\_QUE\_INVALID\_HANDLE is returned.

If the owning process invokes a system semaphore when DosReadQueue or DosPeekQueue is issued, other processes that issue DosWriteQueue must first issue DosOpenSem to access the system semaphore.

# DosWriteQueue — Write to Queue

## C Language

```
#define INCL_DOSQUEUES

USHORT rc = DosWriteQueue(QueueHandle, Request, DataLength, DataBuffer,
                          ElemPriority);

HQUEUE      QueueHandle; /* Queue handle */
USHORT      Request;      /* Request identification data */
USHORT      DataLength;   /* Length of element being added */
PBYTE       DataBuffer;   /* Element being added */
UCHAR       ElemPriority; /* Priority of element being added */

USHORT      rc;           /* return code */
```

## Assembler Language

```
EXTRN DosWriteQueue:FAR
INCL_DOSQUEUES EQU 1

PUSH WORD QueueHandle ;Queue handle
PUSH WORD Request      ;Request identification data
PUSH WORD DataLength   ;Length of element being added
PUSH@ OTHER DataBuffer ;Element being added
PUSH WORD ElemPriority ;Priority of element being added
CALL DosWriteQueue
```

Returns WORD



---

## Chapter 3. Keyboard Function Calls

This chapter reflects the Keyboard API interface of OS/2 only.

For information regarding the keyboard IOCTL interface and keyboard monitor refer to *IBM Operating System/2 Version 1.2 I/O Subsystems And Device Support Volume 1*.

### Notes:

1. Calls marked xPM are not supported by Presentation Manager, and must not be used by Presentation Manager applications. An error code is returned if any of these calls are issued.
2. Calls marked xWPM are not windowable and are not supported by Presentation Manager. They can be used in OS/2 mode.
3. Calls marked FAPI are present in the Family API.
4. Those calls without an icon are:
  - Windowable
  - Presentation Manager
  - Full-screen
  - Not Family API.

---

This call returns a character data record from the keyboard.

**KbdCharIn** (**CharData**, **IOWait**, **KbdHandle**)

## Parameters

**CharData** (*PKBDKEYINFO*) – output

Address of the character data structure:

**asciicharcode** (*UCHAR*)

ASCII character code. The scan code received from the keyboard is translated to the ASCII character code.

**scancode** (*UCHAR*)

Code received from the keyboard. The scan code received from the keyboard is translated to the ASCII character code.

**status** (*UCHAR*)

State of the keystroke event:

Bit	Description
-----	-------------

7 – 6	00 = Undefined
-------	----------------

	01 = Final character, interim character flag off
--	--

	10 = Interim character
--	------------------------

	11 = Final character, interim character flag on.
--	--

5	1 = Immediate conversion requested.
---	-------------------------------------

4 – 2	Reserved.
-------	-----------

1	0 = Scan code is a character.
---	-------------------------------

	1 = Scan code is not a character; is an extended key code from the keyboard.
--	--

0	1 = Shift status returned without character.
---	--

**reserved** (*UCHAR*)

NLS shift status. Reserved, set to zero.

**shiftkeystat** (*USHORT*)

Shift key status.

Bit	Description
-----	-------------

15	SysReq key down
----	-----------------

14	CapsLock key down
----	-------------------

13	NumLock key down
----	------------------

12	ScrollLock key down
----	---------------------

11	Right Alt key down
----	--------------------

10	Right Ctrl key down
----	---------------------

9	Left Alt key down
---	-------------------

8	Left Ctrl key down
---	--------------------

7	Insert on
---	-----------

6	CapsLock on
---	-------------

5	NumLock on
---	------------

4	ScrollLock on
---	---------------

3	Either Alt key down
---	---------------------

2	Either Ctrl key down
---	----------------------

1	Left Shift key down
---	---------------------

0	Right Shift key down
---	----------------------

## KbdCharIn – Read Character, Scan Code

**time (ULONG)**

Time stamp indicating when a key was pressed. It is specified in milliseconds from the time the system was started.

**IOWait (USHORT) – input**

Wait if a character is not available.

Value	Definition
0	Requestor waits for a character if one is not available.
1	Requestor gets an immediate return if no character is available.

**KbdHandle (HKBD) – input**

Default keyboard or the logical keyboard.

**rc (USHORT) – return**

Return code descriptions are:

0	NO_ERROR
375	ERROR_KBD_INVALID_IOWAIT
439	ERROR_KBD_INVALID_HANDLE
445	ERROR_KBD_FOCUS_REQUIRED
447	ERROR_KBD_KEYBOARD_BUSY
464	ERROR_KBD_DETACHED
504	ERROR_KBD_EXTENDED_SG

**Remarks**

On an enhanced keyboard, the secondary enter key returns the normal character 0DH and a scan code of E0H.

Double-byte character codes (DBCS) require two function calls to obtain the entire code.

If shift report is set with KbdSetStatus, the CharData record returned reflects changed shift information only.

Extended ASCII codes are identified with the status byte, bit 1 on and the ASCII character code being either 00H or E0H. Both conditions must be satisfied for the character to be an extended keystroke. For extended ASCII codes, the scan code byte returned is the second code (extended code). Usually the extended ASCII code is the scan code of the primary key that was pressed.

A thread in the foreground session that repeatedly polls the keyboard with KbdCharIn (with no wait), can prevent all regular priority class threads from executing. If polling must be used and a minimal amount of other processing is being performed, the thread should periodically yield to the CPU by issuing a DosSleep call for an interval of at least 5 milliseconds.

**Family API Considerations**

Some options operate differently in the DOS mode than in the OS/2 mode. Therefore, the following restrictions apply to KbdCharIn when coding in the DOS mode:

- The CharData structure includes everything except the time stamp.
- Interim character is not supported
- Status can be 0 or 40H
- KbdHandle is ignored.



# KbdCharIn — Read Character, Scan Code

FAPI xPM

## C Language

```
typedef struct _KBDKEYINFO { /* kbci */
    UCHAR    chChar;        /* ASCII character code */
    UCHAR    chScan;        /* Scan Code */
    UCHAR    fbStatus;      /* State of the character */
    UCHAR    bNlsShift;     /* Reserved (set to zero) */
    USHORT   fsState;       /* State of the shift keys */
    ULONG    time;         /* Time stamp of keystroke (ms since ipl) */
}KBDKEYINFO;

#define INCL_KBD

USHORT rc = KbdCharIn(CharData, IOWait, KbdHandle);

PKBDKEYINFO    CharData;    /* Buffer for data */
USHORT         IOWait;      /* Indicate if wait */
HKBD           KbdHandle;   /* Keyboard handle */

USHORT         rc;          /* return code */
```

## Assembler Language

```
KBDKEYINFO struc
    kbci_chChar    db ? ;ASCII character code
    kbci_chScan    db ? ;Scan Code
    kbci_fbStatus  db ? ;State of the character
    kbci_bNlsShift db ? ;Reserved (set to zero)
    kbci_fsState   dw ? ;state of the shift keys
    kbci_time      dd ? ;time stamp of keystroke (ms since ipl)
KBDKEYINFO ends

EXTRN KbdCharIn:FAR
INCL_KBD EQU 1

PUSH@ OTHER CharData    ;Buffer for data
PUSH  WORD IOWait       ;Indicate if wait
PUSH  WORD KbdHandle     ;Keyboard handle
CALL  KbdCharIn

Returns WORD
```

This call closes the existing logical keyboard identified by the keyboard handle.

**KbdClose (KbdHandle)**

## Parameters

**KbdHandle (HKBD)** – input

Default keyboard or the logical keyboard.

**rc (USHORT)** – return

Return code descriptions are:

<b>0</b>	<b>NO_ERROR</b>
<b>439</b>	<b>ERROR_KBD_INVALID_HANDLE</b>
<b>464</b>	<b>ERROR_KBD_DETACHED</b>
<b>504</b>	<b>ERROR_KBD_EXTENDED_SG</b>

## Remarks

KbdClose blocks while another thread has the keyboard focus (by way of KbdGetFocus) until the thread with the focus issues KbdFreeFocus. Therefore, to prevent KbdClose from blocking, it is recommended that KbdClose be issued only while the current thread has the focus. For example:

<b>KbdGetFocus</b>	Wait until focus available on handle 0.
<b>KbdClose</b>	Close a logical keyboard handle.
<b>KbdClose</b>	Close another logical keyboard handle.
<b>KbdClose</b>	Close still another logical keyboard handle.
<b>KbdFreeFocus</b>	Give up the focus on handle 0.

## C Language

```
#define INCL_KBD
```

```
USHORT rc = KbdClose(KbdHandle);
```

```
HKBD KbdHandle; /* Keyboard handle */
```

```
USHORT rc; /* return code */
```

## Assembler Language

```
EXTRN KbdClose:FAR
```

```
INCL_KBD EQU 1
```

```
PUSH WORD KbdHandle ;Keyboard handle
```

```
CALL KbdClose
```

```
Returns WORD
```

# KbdDeRegister — Deregister Keyboard Subsystem

xWPM

---

This call deregisters a keyboard subsystem previously registered within a session. Only the process that issued the KbdRegister may issue KbdDeRegister.

<b>KbdDeRegister</b> ( )
--------------------------

## Parameters

**rc** (*USHORT*) — return

Return code descriptions are:

<b>0</b>	<b>NO_ERROR</b>
<b>411</b>	<b>ERROR_KBD_DEREGISTER</b>
<b>464</b>	<b>ERROR_KBD_DETACHED</b>
<b>504</b>	<b>ERROR_KBD_EXTENDED_SG</b>

## C Language

```
#define INCL_KBD
```

```
USHORT rc = KbdDeRegister(VOID);
```

```
USHORT rc; /* return code */
```

## Assembler Language

```
EXTRN KbdDeRegister:FAR  
INCL_KBD EQU 1
```

```
CALL KbdDeRegister
```

Returns WORD

This call clears the keystroke buffer.

**KbdFlushBuffer (KbdHandle)**

## Parameters

**KbdHandle** (*HKBD*) — input

Default keyboard or the logical keyboard.

**rc** (*USHORT*) — return

Return code descriptions are:

<b>0</b>	<b>NO_ERROR</b>
<b>439</b>	<b>ERROR_KBD_INVALID_HANDLE</b>
<b>445</b>	<b>ERROR_KBD_FOCUS_REQUIRED</b>
<b>447</b>	<b>ERROR_KBD_KEYBOARD_BUSY</b>
<b>464</b>	<b>ERROR_KBD_DETACHED</b>
<b>504</b>	<b>ERROR_KBD_EXTENDED_SG</b>

## Remarks

KbdFlushBuffer completes when the handle has access to the physical keyboard (focus), or is equal to zero and no other handle has the focus.

## Family API Considerations

Some options operate differently in the DOS mode than in the OS/2 mode. The KbdHandle is ignored when coding in the DOS mode.

## C Language

```
#define INCL_KBD
```

```
USHORT rc = KbdFlushBuffer(KbdHandle);
```

```
HKBD      KbdHandle;    /* Keyboard handle */
```

```
USHORT     rc;          /* return code */
```

## Assembler Language

```
EXTRN KbdFlushBuffer:FAR
```

```
INCL_KBD      EQU 1
```

```
PUSH WORD KbdHandle ;Keyboard handle
```

```
CALL KbdFlushBuffer
```

```
Returns WORD
```

---

This call frees the logical-to-physical keyboard bond created by KbdGetFocus.

<b>KbdFreeFocus</b> (KbdHandle)
---------------------------------

### Parameters

**KbdHandle** (*HKBD*) — input

Default keyboard or the logical keyboard.

**rc** (*USHORT*) — return

Return code descriptions are:

0	NO_ERROR
439	ERROR_KBD_INVALID_HANDLE
445	ERROR_KBD_FOCUS_REQUIRED
464	ERROR_KBD_DETACHED
504	ERROR_KBD_EXTENDED_SG

### Remarks

KbdFreeFocus may be replaced by issuing KbdRegister. Unlike other keyboard subsystem functions, the replaced KbdFreeFocus is called only if there is an outstanding focus.

### C Language

```
#define INCL_KBD

USHORT rc = KbdFreeFocus(KbdHandle);

HKBD      KbdHandle;    /* Keyboard handle */

USHORT     rc;           /* return code */
```

### Assembler Language

```
EXTRN KbdFreeFocus:FAR
INCL_KBD EQU 1

PUSH WORD KbdHandle ;Keyboard handle
CALL KbdFreeFocus
```

Returns WORD

This call allows a process to query the code page being used to translate scan codes to ASCII characters.

**KbdGetCp (Reserved, CodePageID, KbdHandle)**

## Parameters

**Reserved** (*ULONG*) – input

Reserved and must be set to zero.

**CodePageID** (*PUSHORT*) – output

Address of the code page ID located in the application's data area. The keyboard support copies the current code page ID for a specified keyboard handle into this word. The code page ID is equivalent to one of the code page IDs specified in the CONFIG.SYS CODEPAGE = statement or 0000.

**KbdHandle** (*HKBD*) – input

Default keyboard or the logical keyboard.

**rc** (*USHORT*) – return

Return code descriptions are:

<b>0</b>	<b>NO_ERROR</b>
<b>373</b>	<b>ERROR_KBD_PARAMETER</b>
<b>439</b>	<b>ERROR_KBD_INVALID_HANDLE</b>
<b>445</b>	<b>ERROR_KBD_FOCUS_REQUIRED</b>
<b>447</b>	<b>ERROR_KBD_KEYBOARD_BUSY</b>
<b>464</b>	<b>ERROR_KBD_DETACHED</b>
<b>504</b>	<b>ERROR_KBD_EXTENDED_SG</b>

## Remarks

The CodePageID is the currently active keyboard code page. A value of 0 indicates the code page translation table in use is the ROM code page translation table provided by the hardware.

## C Language

```
#define INCL_KBD
```

```
USHORT rc = KbdGetCp(Reserved, CodePageID, KbdHandle);
```

```
ULONG      Reserved;      /* Reserved (must be zero) */
PUSHORT    CodePageID;    /* Code Page ID */
HKBD       KbdHandle;     /* Keyboard handle */

USHORT     rc;            /* return code */
```

## Assembler Language

```
EXTRN KbdGetCp:FAR
INCL_KBD EQU 1

PUSH    DWORD    Reserved      ;Reserved (must be zero)
PUSH    WORD     CodePageID    ;Code Page ID
PUSH    WORD     KbdHandle     ;Keyboard handle
CALL    KbdGetCp
```

Returns WORD

# KbdGetFocus — Get Keyboard Focus

xPM

---

This call binds the logical keyboard to the physical keyboard.

**KbdGetFocus (IOWait, KbdHandle)**

## Parameters

**IOWait** (*USHORT*) — input

Wait if the physical keyboard is already in use by a logical keyboard.

Value	Definition
0	Indicates that the caller wants to wait for the bond.
1	Indicates that the caller does not want to wait for the bond.

**KbdHandle** (*HKBD*) — input

Default keyboard or the logical keyboard.

**rc** (*USHORT*) — return

Return code descriptions are:

0	NO_ERROR
439	ERROR_KBD_INVALID_HANDLE
446	ERROR_KBD_FOCUS_ALREADY_ACTIVE
447	ERROR_KBD_KEYBOARD_BUSY
464	ERROR_KBD_DETACHED
504	ERROR_KDB_EXTENDED_SG

The keyboard handle identifies which logical keyboard to bind to. If the physical keyboard is not bound to a logical or default keyboard, then the bind proceeds immediately. The logical keyboard, identified by the handle, receives all further key strokes from the physical keyboard. If the physical keyboard is already in use by a logical keyboard, then the thread issuing KbdGetFocus waits until the bond can be made. Waiting threads do not execute in any definable order.

## C Language

```
#define INCL_KBD
```

```
USHORT rc = KbdGetFocus(IOWait, KbdHandle);
```

```
USHORT    IOWait;    /* Indicate if wait */  
HKBD      KbdHandle; /* Keyboard handle */
```

```
USHORT    rc;        /* return code */
```

## Assembler Language

```
EXTRN KbdGetFocus:FAR  
INCL_KBD EQU 1
```

```
PUSH WORD IOWait    ;Indicate if wait  
PUSH WORD KbdHandle ;Keyboard handle  
CALL KbdGetFocus
```

Returns WORD

## KbdGetHWId – Query Keyboard Hardware ID

Returns the attached keyboard's hardware-generated Identification value.

**KbdGetHWId (KeyboardID, KbdHandle)**

### Parameters

**KeyboardID (PKBDHWID)** – input

Pointer to the caller's data area where the following structure and data values are:

**length (USHORT)** – input/output

On input, this field should contain the length of the KeyboardID structure. The minimum input length value allowed is 2. On output, this field contains the actual number of bytes returned.

**keybId (USHORT)** – output

OS/2 supported keyboards and their hardware generated IDs are as follows:

ID	Keyboard
0000H	Undetermined keyboard type
0001H	PC-AT Standard Keyboard
AB41H	101 Key Enhanced Keyboard
AB41H	102 Key Enhanced Keyboard
AB54H	88 and 89 Key Enhanced Keyboards
AB85H	122 Key Enhanced Keyboard

**reserved (USHORT)**

Reserved and returned set to zero.

**reserved (USHORT)**

Reserved and returned set to zero.

**KbdHandle (HKBD)** – input

Word identifying the logical keyboard.

**rc (USHORT)** – return

Return code descriptions are:

373	ERROR_KBD_PARAMETER
447	ERROR_KBD_KEYBOARD_BUSY
464	ERROR_KBD_DETACHED
504	ERROR_KBD_EXTENDED_SG

### Remarks

In past OS/2 releases, all keyboards could be supported by knowing the hardware family information available with keyboard IOCTL 77H. However, with the addition of the 122-key keyboard, recognition was not containable by hardware family information alone. The 122-key keyboard has a number of differences from other keyboards. Therefore, applications performing keystroke specific functions may need to determine specifically which keyboard is attached.

This function is of particular usefulness for applications providing Custom Translate Tables and mapping keyboard layouts.



# KbdGetHWId — Query Keyboard Hardware ID

## C Language

```
typedef struct _KBDHWID {
    USHORT length;           /* length in bytes of this structure */
    USHORT kbd_id;           /* attached keyboard's hardware ID (returned) */
    USHORT reserved1;        /* reserved (set to zero) */
    USHORT reserved2;        /* reserved (set to zero) */
}KBDHWID;

#define INCL_KBD

USHORT rc = KbdGetHWId(KeyboardID, KbdHandle);

PKBDHWID      KeyboardID;    /* Keyboard ID structure (returned) */
HKBD          KbdHandle;     /* Keyboard handle */

USHORT        rc;            /* return code */
```

## Assembler Language

```
KBDHWID struc
    length;          dw ? ;length in bytes of this structure
    kbd_id;          dw ? ;attached keyboard's hardware ID (returned)
    reserved1;       dw ? ;reserved (set to zero)
    reserved2;       dw ? ;reserved (set to zero)
KBDHWID ends

EXTRN KbdGetHWId:FAR
INCL_KBD      EQU 1

PUSH@ OTHER KeyboardID    ;Keyboard ID structure (returned)
PUSH WORD KbdHandle       ;Keyboard handle
CALL KbdGetHWId

Returns WORD
```

This call gets the current state of the keyboard.

**KbdGetStatus (StatData, KbdHandle)**

## Parameters

**StatData** (*PKBDINFO*) — output

Address of the keyboard status structure:

**length** (*USHORT*)

Length, in bytes, of this data structure, including length.

**10** Only valid value.

**sysstate** (*USHORT*)

State as follows:

Bit	Description
<b>15 – 9</b>	Reserved, set to zero.
<b>8</b>	Shift return is on.
<b>7</b>	Length of the turn-around character (meaningful only if bit 6 is on).
<b>6</b>	Turn-around character is modified.
<b>5</b>	Interim character flags are modified.
<b>4</b>	Shift state is modified.
<b>3</b>	ASCII mode is on.
<b>2</b>	Binary mode is on.
<b>1</b>	Echo off.
<b>0</b>	Echo on.

**turnchardef** (*USHORT*)

Definition of the turn-around character. In ASCII and extended-ASCII format, the turn-around character is defined as the carriage return. In ASCII format only, the turn-around character is defined in the low-order byte.

**Intcharflag** (*USHORT*)

Interim character flags:

Bit	Description
<b>15 – 8</b>	NLS shift state.
<b>7</b>	Interim character flag is on.
<b>6</b>	Reserved, set to zero.
<b>5</b>	Application requested immediate conversion.
<b>4 – 0</b>	Reserved, set to zero.

**shiftstate** (*USHORT*)

Shift state as follows:

Bit	Description
<b>15</b>	SysReq key down
<b>14</b>	CapsLock key down
<b>13</b>	NumLock key down
<b>12</b>	ScrollLock key down
<b>11</b>	Right Alt key down
<b>10</b>	Right Ctrl key down
<b>9</b>	Left Alt key down
<b>8</b>	Left Ctrl key down
<b>7</b>	Insert on
<b>6</b>	CapsLock on
<b>5</b>	NumLock on
<b>4</b>	ScrollLock on

# KbdGetStatus — Get Keyboard Status

FAPI xPM

3	Either Alt key down
2	Either Ctrl key down
1	Left Shift key down
0	Right Shift key down.

**KbdHandle** (*HKBD*) — input  
Default keyboard or the logical keyboard.

**rc** (*USHORT*) — return  
Return code descriptions are:

0	NO_ERROR
376	ERROR_KBD_INVALID_LENGTH
439	ERROR_KBD_INVALID_HANDLE
445	ERROR_KBD_FOCUS_REQUIRED
447	ERROR_KBD_KEYBOARD_BUSY
464	ERROR_KBD_DETACHED
504	ERROR_KBD_EXTENDED_SG

## Remarks

The initial state of the keyboard is established by the system at application load time. Some default states may be modified by the application through KbdSetStatus. KbdGetStatus returns only those keyboard parameters initially set by KbdSetStatus. The returned parameters are:

- Input Mode
- Interim Character Flags
- Shift State
- Echo State
- Turnaround Character

KbdGetStatus completes only when the handle has access to the physical keyboard (focus) or the handle is 0 and no other handle has the focus.

## Family API Considerations

Some options operate differently in the DOS mode than in the OS/2 mode. Therefore, the following restrictions apply to KbdGetStatus when coding in the DOS mode:

- Interim character is not supported
- TurnAround character is not supported
- NLS\_SHIFT\_STATE is always NULL.
- KbdHandle is ignored.

## C Language

```
typedef struct _KBDINFO {      /* kbst */
    USHORT cb;                 /* length in bytes of this structure */
    USHORT fsMask;              /* bit mask of functions to be altered */
    USHORT chTurnAround;        /* define TurnAround character */
    USHORT fsInterim;           /* interim character flags */
    USHORT fsState;             /* shift states */
}KBDINFO;

#define INCL_KBD

USHORT rc = KbdGetStatus(Structure, KbdHandle);

PKBDINFO      Structure;      /* Data structure */
HKBD          KbdHandle;      /* Keyboard handle */

USHORT        rc;             /* return code */
```

**Assembler Language**

```
KBDINFO struc
    kbst_cb          dw ? ;length in bytes of this structure
    kbst_fsMask      dw ? ;bit mask of functions to be altered
    kbst_chTurnAround dw ? ;define TurnAround character
    kbst_fsInterim   dw ? ;interim character flags
    kbst_fsState     dw ? ;shift states
KBDINFO ends
```

```
EXTRN KbdGetStatus:FAR
INCL_KBD EQU 1
```

```
PUSH@ OTHER Structure ;Data structure
PUSH WORD KbdHandle ;Keyboard handle
CALL KbdGetStatus
```

Returns WORD

# KbdOpen — Open a Logical Keyboard

xPM

---

This call creates a new logical keyboard.

<b>KbdOpen (KbdHandle)</b>
----------------------------

## Parameters

**KbdHandle (PHKBD)** — output  
Address of the logical keyboard.

**rc (USHORT)** — return  
Return code descriptions are:

<b>0</b>	<b>NO_ERROR</b>
<b>440</b>	<b>ERROR_KBD_NO_MORE_HANDLE</b>
<b>441</b>	<b>ERROR_KBD_CANNOT_CREATE_KCB</b>
<b>464</b>	<b>ERROR_KBD_DETACHED</b>
<b>504</b>	<b>ERROR_KBD_EXTENDED_SG</b>

## Remarks

KbdOpen blocks while another thread has the keyboard focus (by way of KbdGetFocus) until the thread with the focus issues KbdFreeFocus. Therefore, to prevent KbdOpen from blocking, it is recommended that KbdOpen be issued only while the current thread has the focus. For example:

<b>KbdGetFocus</b>	wait until focus available on handle 0
<b>KbdOpen</b>	get a logical keyboard handle
<b>KbdOpen</b>	get another logical keyboard handle
<b>KbdOpen</b>	get yet another logical keyboard handle
<b>KbdFreeFocus</b>	give up the focus on handle 0.

## C Language

```
#define INCL_KBD

USHORT rc = KbdOpen(KbdHandle);

PHKBD KbdHandle; /* Keyboard handle */

USHORT rc; /* return code */
```

## Assembler Language

```
EXTRN KbdOpen:FAR
INCL_KBD EQU 1

PUSH@ WORD KbdHandle ;Keyboard handle
CALL KbdOpen

Returns WORD
```

This call returns any available character data record from the keyboard without removing it from the buffer.

**KbdPeek (CharData, KbdHandle)**

## Parameters

**CharData (PKBDKEYINFO)** – output

Address of the character data information:

**asciicharcode (UCHAR)**

ASCII character code. The scan code received from the keyboard is translated to the ASCII character code.

**scancode (UCHAR)**

Code received from the keyboard hardware.

**status (UCHAR)**

State of the keystroke event:

Bit	Description
7 – 6	00 = Undefined. 01 = Final character, interim character flag off. 10 = Interim character. 11 = Final character, interim character flag on.
5	1 = Immediate conversion requested.
4 – 2	Reserved, set to zero.
1	0 = Scan code is a character. 1 = Scan code is not a character; it is an extended key code from the keyboard.
0	1 = Shift status returned without character.

**reserved (UCHAR)**

NLS shift status. Reserved, set to zero.

**shiftkeystat (USHORT)**

Shift key status.

Bit	Description
15	SysReq key down
14	CapsLock key down
13	NumLock key down
12	ScrollLock key down
11	Right Alt key down
10	Right Ctrl key down
9	Left Alt key down
8	Left Ctrl key down
7	Insert on
6	CapsLock on
5	NumLock on
4	ScrollLock on
3	Either Alt key down
2	Either Ctrl key down
1	Left Shift key down
0	Right Shift key down

# KbdPeek — Peek at Character, Scan Code

FAPI xPM

**time** (*ULONG*)

Time stamp indicating when a key was pressed. It is specified in milliseconds from the time the system was started.

**KbdHandle** (*HKBD*) — input

Default keyboard or the logical keyboard.

**rc** (*USHORT*) — return

Return code descriptions are:

0	NO_ERROR
439	ERROR_KBD_INVALID_HANDLE
445	ERROR_KBD_FOCUS_REQUIRED
447	ERROR_KBD_KEYBOARD_BUSY
464	ERROR_KBD_DETACHED
504	ERROR_KBD_EXTENDED_SG

## Remarks

On an enhanced keyboard, the secondary enter key returns the normal character 0DH and a scan code of E0H.

Double-byte character codes (DBCS) require two function calls to obtain the entire code.

If shift report is set with KbdSetStatus the CharData record returned, reflects changed shift information only.

Extended ASCII codes are identified with the status byte, bit 1 on and the ASCII character code being either 00H or E0H. Both conditions must be satisfied for the character to be an extended keystroke. For extended ASCII codes, the scan code byte returned is the second code (extended code). Usually the extended ASCII code is the scan code of the primary key that was pressed.

A thread in the foreground session that repeatedly polls the keyboard with KbdCharIn (with no wait), can prevent all regular priority class threads from executing. If polling must be used and a minimal amount of other processing is being performed, the thread should periodically yield the CPU by issuing a DosSleep call for an interval of at least 5 milliseconds.

## Family API Considerations

Some options operate differently in the DOS mode than in the OS/2 mode. Therefore, the following restrictions apply to KbdPeek when coding for the DOS mode:

- The CharData structure includes everything except the time stamp.
- Interim character is not supported.
- Status can be 0 or 1.
- KbdHandle is ignored.

## C Language

```
typedef struct _KBDKEYINFO {    /* kbci */
    UCHAR    chChar;           /* ASCII character code */
    UCHAR    chScan;          /* Scan Code */
    UCHAR    fbStatus;         /* State of the character */
    UCHAR    bNlsShift;        /* Reserved (set to zero) */
    USHORT   fsState;          /* State of the shift keys */
    ULONG    time;             /* Time stamp of keystroke (ms since ipl) */
}KBDKEYINFO;

#define INCL_KBD

USHORT rc = KbdPeek(CharData, KbdHandle);

PKBDKEYINFO    CharData;    /* Buffer for data */
HKBD           KbdHandle;   /* Keyboard handle */

USHORT         rc;          /* return code */
```

## Assembler Language

```
KBDKEYINFO struc
    kbc_i_chChar    db  ? ;ASCII character code
    kbc_i_chScan    db  ? ;Scan Code
    kbc_i_fbStatus  db  ? ;State of the character
    kbc_i_bNlsShift db  ? ;Reserved (set to zero)
    kbc_i_fsState   dw  ? ;state of the shift keys
    kbc_i_time      dd  ? ;time stamp of keystroke (ms since ipl)
KBDKEYINFO ends
```

```
EXTRN KbdPeek:FAR
INCL_KBD      EQU 1
```

```
PUSH@ OTHER CharData    ;Buffer for data
PUSH WORD KbdHandle     ;Keyboard handle
CALL KbdPeek
```

```
Returns WORD
```



# KbdRegister — Register Keyboard Subsystem

xWPM

This call registers a keyboard subsystem within a session.

**KbdRegister (ModuleName, EntryPoint, FunctionMask)**

## Parameters

**ModuleName** (*PSZ*) — input

Address of the dynamic link module name. Maximum length is 9 bytes (including ASCIIZ terminator).

**EntryPoint** (*PSZ*) — input

Address of the dynamic link entry point name of a routine that receives control when any of the registered functions are called. Maximum length is 33 bytes (including ASCIIZ terminator).

**FunctionMask** (*ULONG*) — input

A bit mask where each bit identifies a keyboard function being registered. The bit values are:

Bit	Description
31 – 15	Reserved, must be set to zero.
14	KbdGetHWId
13	KbdSetCustXt
12	KbdXlate
11	KbdSetCp
10	KbdGetCp
9	KbdFreeFocus
8	KbdGetFocus
7	KbdClose
6	KbdOpen
5	KbdStringIn
4	KbdSetStatus
3	KbdGetStatus
2	KbdFlushBuffer
1	KbdPeek
0	KbdCharIn

**rc** (*USHORT*) — return

Return code descriptions are:

0	NO_ERROR
408	ERROR_KBD_INVALID_ASCII
409	ERROR_KBD_INVALID_MASK
410	ERROR_KBD_REGISTER
464	ERROR_KBD_DETACHED
504	ERROR_KBD_EXTENDED_SG

## Remarks

There can be only one KbdRegister call outstanding for each session without an intervening KbdDeRegister. KbdDeRegister must be issued by the same process that issued the KbdRegister.

## C Language

```
#define INCL_KBD
```

```
USHORT rc = KbdRegister(ModuleName, EntryPoint, FunctionMask);
```

```
PSZ      ModuleName; /* Module name */
PSZ      EntryPoint; /* Entry point name */
ULONG    FunctionMask; /* Function mask */

USHORT    rc; /* return code */
```

## KbdRegister – Register Keyboard Subsystem

### Assembler Language

```
EXTRN KbdRegister:FAR
INCL_KBD EQU 1

PUSH@ ASCIIZ ModuleName ;Module name
PUSH@ ASCIIZ EntryPoint ;Entry point name
PUSH DWORD FunctionMask ;Function mask
CALL KbdRegister

Returns WORD
```

# KbdSetCp – Set the Code Page

xPM

This call allows the process to set the code page used to translate key strokes received from the keyboard.

**KbdSetCp (Reserved, CodePageID, KbdHandle)**

## Parameters

**Reserved (USHORT)** – input  
Reserved and must be set to zero.

**CodePageID (USHORT)** – input  
CodePageID represents a code-page ID in the application's data area. The code-page ID must be equivalent to one of the code-page IDs specified on the CONFIG.SYS CODEPAGE = statement or 0000. If the code-page ID does not match one of the IDs on the CODEPAGE = statement, an error results. The code-page word must have one of these code-page identifiers:

Identifier	Description
0	Resident code page
437	IBM PC US 437
850	Multilingual
860	Portuguese
863	Canadian-French
865	Nordic.

**KbdHandle (HKBD)** – input  
Default keyboard or the logical keyboard.

**rc (USHORT)** – return  
Return code descriptions are:

0	NO_ERROR
439	ERROR_KBD_INVALID_HANDLE
445	ERROR_KBD_FOCUS_REQUIRED
447	ERROR_KBD_KEYBOARD_BUSY
448	ERROR_KBD_INVALID_CODEPAGE
464	ERROR_KBD_DETACHED
504	ERROR_KBD_EXTENDED_SG

## Remarks

Keyboard code page support is not available without the DEVINFO=KBD statement in the CONFIG.SYS file. Refer to the *IBM Operating System/2 Using the System* for a complete description of CODEPAGE and DEVINFO.

## C Language

```
#define INCL_KBD

USHORT rc = KbdSetCp(Reserved, CodePageID, KbdHandle);

USHORT Reserved; /* Reserved (must be zero) */
USHORT CodePageID; /* code page ID */
HKBD KbdHandle; /* Keyboard handle */

USHORT rc; /* return code */
```

**Assembler Language**

```
EXTRN KbdSetCp:FAR
INCL_KBD EQU 1

PUSH WORD Reserved ;Reserved (must be zero)
PUSH WORD CodePageID ;code page ID
PUSH WORD KbdHandle ;Keyboard handle
CALL KbdSetCp
```

Returns WORD

# KbdSetCustXt —

## Set Custom Translate Table

xPM

This call installs, on the specified handle, the translate table which this call points to. This translate table affects only this handle.

**KbdSetCustXt** (Xlatetable, KbdHandle)

### Parameters

**Xlatetable** (*PUSHORT*) — input

A pointer to the translation table used to translate scan code to ASCII code for a specified handle. The format of the translate table is documented in the Set Code Page IOCTL 50H. Refer to *IBM Operating System/2 Version 1.2 I/O Subsystems And Device Support Volume 1* for a complete discussion of Set Code Page IOCTL 50H.

**KbdHandle** (*HKBD*) — input

Default keyboard or the logical keyboard.

**rc** (*USHORT*) — return

Return code descriptions are:

0	NO_ERROR
377	ERROR_KBD_INVALID_ECHO_MASK
378	ERROR_KBD_INVALID_INPUT_MASK
439	ERROR_KBD_INVALID_HANDLE
445	ERROR_KBD_FOCUS_REQUIRED
447	ERROR_KBD_KEYBOARD_BUSY
464	ERROR_KBD_DETACHED
504	ERROR_KBD_EXTENDED_SG

### Remarks

The translate table must be maintained in the caller's memory. No copy of the translate table is made by KbdSetCustXt.

KbdSetCp reverses the action of KbdSetCustXt and sets the handle equal to one of the system translate tables. If memory is dynamically allocated by the caller for the translate table and is freed before the KbdSetCp is performed, KbdSetCp and future translations may fail.

### C Language

```
#define INCL_KBD

USHORT rc = KbdSetCustXt(Xlatetable, KbdHandle);

PUSHORT      Xlatetable;    /* Translation Table */
HKBD         KbdHandle;     /* Keyboard handle */

USHORT      rc;              /* return code */
```

### Assembler Language

```
EXTRN KbdSetCustXt:FAR
INCL_KBD EQU 1

PUSH@ WORD CodePage ;Translation Table
PUSH WORD KbdHandle ;Keyboard handle
CALL KbdSetCustXt
```

Returns WORD

# KbdSetFgnd – Set Foreground Keyboard Priority

---

This call raises the priority of the foreground keyboard's thread.

**KbdSetFgnd ( )**

## Parameters

**rc** (*USHORT*) – return

Return code descriptions are:

<b>0</b>	<b>NO_ERROR</b>
<b>447</b>	<b>ERROR_KBD_KEYBOARD_BUSY</b>
<b>504</b>	<b>ERROR_KBD_EXTENDED_SG</b>

## Remarks

KbdSetFgnd marks the current process that owns the keyboard. Threads in this process receive a priority boost. The previous foreground keyboard threads lose their priority boost.

This function should only be issued by a Keyboard Subsystem during KbdCharIn or KbdStringIn processing.

## C Language

```
#define INCL_KBD
```

```
USHORT rc = KbdSetFgnd(VOID);
```

```
USHORT rc; /* return code */
```

## Assembler Language

```
EXTRN KbdSetFgnd:FAR
```

```
INCL_KBD EQU 1
```

```
CALL KbdSetFgnd
```

```
Returns WORD
```

# KbdSetStatus – Set Keyboard Status

FAPI xPM

This call sets the characteristics of the keyboard.

**KbdSetStatus (StatData, KbdHandle)**

## Parameters

**StatData (PKBDINFO)** – input

Address of the keyboard status structure:

**length (USHORT)**

Length, in bytes, of this data structure, including length.

**10** Only valid value.

**sysstate (USHORT)**

The system state altered by this call. If bits 0 and 1 are off, the echo state of the system is not altered. If bits 2 and 3 are off, the binary and ASCII state of the system is not altered. If bits 0 and 1 are on, or if bits 2 and 3 are on, the function returns an error. If binary mode is set, echo is ignored.

Bit	Description
15 – 9	Reserved, set to zero
8	Shift return is on
7	Length of the turn-around character (meaningful only if bit 6 is on).
6	Turn-around character is modified
5	Interim character flags are modified
4	Shift state is modified
3	ASCII mode is on
2	Binary mode is on
1	Echo off
0	Echo on

**turnchardef (USHORT)**

Definition of the turn-around character. In ASCII and extended-ASCII format, the turn-around character is defined as the carriage return. In ASCII format only, the turn-around character is defined in the low-order byte.

**intcharflag (USHORT)**

Interim character flags:

Bit	Description
15 – 8	NLS shift state.
7	Interim character flag is on
6	Reserved, set to zero
5	Application requested immediate conversion
4 – 0	Reserved, set to zero

**shiftstate (USHORT)**

Shift state.

Bit	Description
15	SysReq key down
14	CapsLock key down
13	NumLock key down
12	ScrollLock key down
11	Right Alt key down
10	Right Ctrl key down
9	Left Alt key down
8	Left Ctrl key down
7	Insert on

6	CapsLock on
5	NumLock on
4	ScrollLock on
3	Either Alt key down
2	Either Ctrl key down
1	Left Shift key down
0	Right Shift key down

**KbdHandle** (*HKBD*) — input

Default keyboard or the logical keyboard.

**rc** (*USHORT*) — return

Return code descriptions are:

0	NO_ERROR
376	ERROR_KBD_INVALID_LENGTH
377	ERROR_KBD_INVALID_ECHO_MASK
378	ERROR_KBD_INVALID_INPUT_MASK
439	ERROR_KBD_INVALID_HANDLE
445	ERROR_KBD_FOCUS_REQUIRED
447	ERROR_KBD_KEYBOARD_BUSY
464	ERROR_KBD_DETACHED
504	ERROR_KBD_EXTENDED_SG

## Remarks

Shift return (bit 8 in sysstate) must be disabled in ASCII mode.

KbdSetStatus is ignored for a Vio-windowed application.

## Family API Considerations

Some options operate differently in the DOS mode than in the OS/2 mode. Therefore, the following restrictions apply to KbdSetStatus when coding in the DOS mode:

- KbdSetStatus does not accept a bit mask of 10 (ASCII on, Echo Off).
- Raw (binary) Mode and Echo On are not supported and return an error if requested.
- KbdHandle is ignored.
- Interim character is not supported.
- Turnaround character is not supported.

## C Language

```
typedef struct _KBDINFO {      /* kbst */
    USHORT cb;                 /* length in bytes of this structure */
    USHORT fsMask;              /* bit mask of functions to be altered */
    USHORT chTurnAround;        /* define TurnAround character */
    USHORT fsInterim;           /* interim character flags */
    USHORT fsState;             /* shift states */
}KBDINFO;

#define INCL_KBD

USHORT rc = KbdSetStatus(Structure, KbdHandle);

PKBDINFO      Structure;      /* Data structure */
HKBD           KbdHandle;     /* Keyboard Handle */

USHORT         rc;             /* return code */
```



# KbdSetStatus — Set Keyboard Status

FAPI xPM

## Assembler Language

```
KBDINFO struc
    kbst_cb          dw ? ;length in bytes of this structure
    kbst_fsMask      dw ? ;bit mask of functions to be altered
    kbst_chTurnAround dw ? ;define TurnAround character
    kbst_fsInterim   dw ? ;interim character flags
    kbst_fsState      dw ? ;shift states
KBDINFO ends
```

```
EXTRN KbdSetStatus:FAR
INCL_KBD EQU 1
```

```
PUSH@ OTHER Structure ;Data structure
PUSH WORD KbdHandle ;Keyboard Handle
CALL KbdSetStatus
```

Returns WORD

---

This call reads a character string (character codes only) from the keyboard.

**KbdStringIn** (*CharBuffer*, *StringLength*, *IOWait*, *KbdHandle*)

## Parameters

**CharBuffer** (*PCH*) – output

Address of the character string buffer.

**StringLength** (*PSTRINGINBUF*) – input/output

Address of the length of the character string buffer. On entry, *buflen* is the maximum length, in bytes, of the buffer. The maximum length that can be specified is 255. Template processing has meaning only in the ASCII mode.

**buflen** (*USHORT*)

Length of the input buffer.

**inputlen** (*USHORT*)

Number of bytes read into the buffer.

**IOWait** (*USHORT*) – input

Wait if a character is not available.

Value	Definition
0	Wait. In Binary input mode, the requestor waits until <i>CharBuffer</i> is full. In ASCII input mode, the requestor waits until a carriage return is pressed.
1	No wait. The requestor gets an immediate return if no characters are available. If characters are available, <i>KbdStringIn</i> returns immediately with as many characters as are available (up to the maximum). No wait is not supported in ASCII input mode.

**KbdHandle** (*HKBD*) – input

Default keyboard or the logical keyboard.

**rc** (*USHORT*) – return

Return code descriptions are:

0	NO_ERROR
375	ERROR_KBD_INVALID_IOWAIT
439	ERROR_KBD_INVALID_HANDLE
445	ERROR_KBD_FOCUS_REQUIRED
464	ERROR_KBD_DETACHED
504	ERROR_KBD_EXTENDED_SG

## Remarks

The character strings may be optionally echoed on the display if echo mode is set. When echo is on each character is echoed as it is read from the keyboard. Echo mode and BINARY mode are mutually exclusive. Reference *KbdSetStatus* and *KbdGetStatus* for more information.

The default input mode is ASCII. In ASCII mode, 2-byte character codes only return in complete form. An extended ASCII code is returned in a 2-byte string. The first byte is 0DH or E0H and the next byte is an extended code.

In input mode (BINARY, ASCII), The following returns can be set and retrieved with *KbdSetStatus* and *KbdGetStatus*:

- Turnaround Character
- Echo Mode
- Interim Character Flag
- Shift State

# KbdStringIn —

## Read Character String

FAPI xPM

The received input length is also used by the KbdStringIn line edit functions for re-displaying and entering a caller specified string. On the next KbdStringIn call the received input length indicates the length of the input buffer that may be recalled by the user using the line editing keys. A value of 0 inhibits the line editing function for the current KbdStringIn request.

KbdStringIn completes when the handle has access to the physical keyboard (focus), or is equal to zero and no other handle has the focus.

### Family API Considerations

Some options operate differently in the DOS mode than in the OS/2 mode. Therefore, the following restrictions apply to KbdStringIn when coding in the DOS mode:

- KbdHandle is ignored

Refer to the DosRead Family API Considerations for differences between DOS and OS/2 mode when reading from a handle opened to the CON device.

### C Language

```
typedef struct _STRINGINBUF {    /* kbsi */
    USHORT cb;                  /* input buffer length */
    USHORT cchIn;               /* received input length */
} STRINGINBUF;

#define INCL_KBD

USHORT rc = KbdStringIn(CharBuffer, Length, IOWait, KbdHandle);

PCH          CharBuffer;    /* Char string buffer */
PSTRINGINBUF Length;        /* Length table */
USHORT       IOWait;        /* Indicate if wait for char */
HKBD         KbdHandle;     /* Keyboard handle */

USHORT       rc;            /* return code */
```

### Assembler Language

```
STRINGINBUF struc
    kbsi_cb    dw ? ;input buffer length
    kbsi_cchIn dw ? ;received input length
STRINGINBUF ends

EXTRN KbdStringIn:FAR
INCL_KBD      EQU 1

PUSH@ OTHER CharBuffer ;Char string buffer
PUSH@ OTHER Length     ;Length table
PUSH  WORD IOWait       ;Indicate if wait for char
PUSH  WORD KbdHandle    ;Keyboard handle
CALL  KbdStringIn

Returns WORD
```

This call synchronizes access from a keyboard subsystem to the keyboard device driver.

**KbdSynch (IOWait)**

## Parameters

**IOWait (USHORT)** – input

Wait for the bond. Values are:

Value	Definition
0	Indicates the requestor does not wait for access to the device driver.
1	Indicates the requestor waits for access to the device driver.

**rc (USHORT)** – return

Return code descriptions are:

0	NO_ERROR
121	ERROR_SEM_TIMEOUT

## Remarks

KbdSynch blocks all other threads within a session until return from the subsystem to the router. To ensure proper synchronization, KbdSynch should be issued by a keyboard subsystem if it intends to issue a DosDevIOCtl or access dynamically shared data. KbdSynch does not protect globally shared data from threads in other sessions.

## C Language

```
#define INCL_KBD
```

```
USHORT rc = KbdSynch(IOWait);
```

```
USHORT      IOWait;      /* Indicate if wait */
```

```
USHORT      rc;          /* return code */
```

## Assembler Language

```
EXTRN KbdSynch:FAR
```

```
INCL_KBD      EQU 1
```

```
PUSH WORD IOWait      ;Indicate if wait
```

```
CALL KbdSynch
```

```
Returns WORD
```

---

This call translates scan codes with shift states into ASCII codes.

**KbdXlate (XlateRecord, KbdHandle)**

## Parameters

**XlateRecord (PKBDTRANS)** – input

Address of the translation record structure:

**chardata (KBDKEYINFO)**

Character data information structure as defined in KbdCharIn call.

**kbdflag (USHORT)**

See the KbdDDFlagWord call in the "Keyboard Device Driver" section of *IBM Operating System/2 Version 1.2 I/O Subsystems And Device Support Volume 1*.

**xlate (USHORT)**

Translation flag:

Value	Definition
0	Translation incomplete.
1	Translation complete.

**xlatestate1 (USHORT)**

Identifies the state of translation across successive calls; initially the value should be zero. It may take several calls to this function to complete a character. The value should not be changed unless a new translation is required, that is, reset value to zero.

**xlatestate2 (USHORT)**

See description for xlatestate1.

**KbdHandle (HKBD)** – input

Default keyboard or the logical keyboard.

**rc (USHORT)** – return

Return code descriptions are:

0	NO_ERROR
439	ERROR_KBD_INVALID_HANDLE
445	ERROR_KBD_FOCUS_REQUIRED
447	ERROR_KBD_KEYBOARD_BUSY
464	ERROR_KBD_DETACHED
504	ERROR_KBD_EXTENDED_SG

## Remarks

It may take several calls to complete a translation because of accent key combinations, or other complex operations.

The Xlatestate1 and Xlatestate2 are for use by the keyboard translation routines. These fields are reserved and must only be accessed by the caller prior to starting a translation sequence and then they must be set to zero. The KbdXlate function is intended to be used for translating a particular scan code for a given shift state. The KbdXlate function is not intended to be a replacement for the OS/2 system keystroke translation function.

## C Language

```
typedef struct _KBDTRANS { /* kbx1 */
    UCHAR    chChar; /* ASCII character code */
    UCHAR    chScan; /* Scan code */
    UCHAR    fbStatus; /* State of the character */
    UCHAR    bNlsShift; /* Shift status (reserved set to zero) */
    USHORT   fsState; /* Shift state */
    ULONG    time;
    USHORT   fsDD;
    USHORT   fsXlate;
    USHORT   fsShift;
    USHORT   sZero;
} KBDTRANS;

#define INCL_KBD

USHORT rc = KbdXlate(XlateRecord, KbdHandle);

PKBDTRANS    XlateRecord; /* Translation Record */
HKBD         KbdHandle; /* Keyboard handle */

USHORT       rc; /* return code */
```

## Assembler Language

```
KBDTRANS struc
    kbx1_chChar    db ? ;ASCII character code
    kbx1_chScan    db ? ;scan code
    kbx1_fbStatus  db ? ;State of the character
    kbx1_bNlsShift db ? ;shift status (reserved set to zero)
    kbx1_fsState   dw ? ;shift state
    kbx1_time      dd ?
    kbx1_fsDD      dw ?
    kbx1_fsXlate   dw ?
    kbx1_fsShift   dw ?
    kbx1_sZero     dw ?
KBDTRANS ends

EXTRN KbdXlate:FAR
INCL_KBD EQU 1

PUSH@ OTHER XlateRecord ;Translation Record
PUSH WORD KbdHandle ;Keyboard handle
CALL KbdXlate

Returns WORD
```



---

## Chapter 4. Mouse Function Calls

This chapter reflects the Mouse API interface of OS/2 only.

For information regarding mouse device drivers, mouse pointer draw device, mouse installation and mouse IOCTLs, refer to *IBM Operating System/2 Version 1.2 I/O Subsystems And Device Support Volume 1*.

### Notes:

1. Calls marked xPM are not supported by Presentation Manager, and must not be used by Presentation Manager applications. An error code is returned if any of these calls are issued.
2. Calls marked xWPM are not windowable and are not supported by Presentation Manager. They can be used in OS/2 mode.
3. Calls marked FAPI are present in the Family API.
4. Those calls without an icon are:
  - Windowable
  - Presentation Manager
  - Full-screen
  - Not Family API.



# MouClose —

## Close Mouse Device

xPM

---

This call closes the mouse device for the current session.

<b>MouClose</b> ( <i>DeviceHandle</i> )
---

### Parameters

**DeviceHandle** (*HMOU*) — input  
Mouse device handle from a previous **MouOpen**.

**rc** (*USHORT*) — return  
Return code descriptions are:

0	NO_ERROR
385	ERROR_MOUSE_NO_DEVICE
466	ERROR_MOU_DETACHED
501	ERROR_MOUSE_NO_CONSOLE
505	ERROR_MOU_EXTENDED_SG

### Remarks

**MouClose** closes the mouse device for the current session and removes the mouse device driver handle from the list of valid open mouse device handles.

### C Language

```
#define INCL_MOU

USHORT rc = MouClose(DeviceHandle);

HMOU      DeviceHandle; /* Mouse device handle */

USHORT    rc;           /* return code */
```

### Assembler Language

```
EXTRN MouClose:FAR
INCL_MOU      EQU 1

PUSH WORD DeviceHandle ;Mouse device handle
CALL MouClose

Returns WORD
```

---

This call deregisters a mouse subsystem previously registered within a session.

<b>MouDeRegister ( )</b>
--------------------------

## Parameters

**rc** (*USHORT*) – return

Return code descriptions are:

<b>0</b>	<b>NO_ERROR</b>
<b>385</b>	<b>ERROR_MOUSE_NO_DEVICE</b>
<b>416</b>	<b>ERROR_MOUSE_DEREGISTER</b>
<b>466</b>	<b>ERROR_MOU_DETACHED</b>
<b>505</b>	<b>ERROR_MOU_EXTENDED_SG</b>

## Remarks

Processes issuing MouDeRegister calls must conform to the following rules:

- The process that issued the MouRegister must release the session (by a MouDeRegister) from the registered subsystem before another PID may issue MouRegister.
- The process that issued the MouRegister is the only process that may issue MouDeRegister against the currently registered subsystem.
- After the owning process has released the subsystem with a MouDeRegister, any other process in the session may issue a MouRegister and therefore modify the mouse support for the entire session.

## C Language

```
#define INCL_MOUSE

USHORT rc = MouDeRegister(VOID);

USHORT rc; /* return code */
```

## Assembler Language

```
EXTRN MouDeRegister:FAR
INCL_MOUSE EQU 1

CALL MouDeRegister

Returns WORD
```

---

This call allows a process to notify the mouse device driver that an area previously restricted to the pointer image is now available to the mouse device driver.

**MouDrawPtr (DeviceHandle)**

## Parameters

**DeviceHandle (HMOU)** – input

Mouse device handle from a previous MouOpen.

**rc (USHORT)** – return

Return code descriptions are:

<b>0</b>	<b>NO_ERROR</b>
<b>385</b>	<b>ERROR_MOUSE_NO_DEVICE</b>
<b>466</b>	<b>ERROR_MOU_DETACHED</b>
<b>501</b>	<b>ERROR_MOUSE_NO_CONSOLE</b>
<b>505</b>	<b>ERROR_MOU_EXTENDED_SG</b>

## Remarks

The collision area (the pointer image restricted area) is established by MouOpen and by MouRemovePtr. MouDrawPtr nullifies the effect of the MouRemovePtr command. If there was no previous MouDrawPtr command or if a previous MouDrawPtr command has already nullified the collision area, the MouRemovePtr command is effectively a null operation.

This call is required to begin session pointer image drawing. Immediately after MouOpen is issued, the collision area is defined as the size of the display. A MouDrawPtr is issued to begin pointer drawing after the MouOpen.

## C Language

```
#define INCL_MOU
```

```
USHORT rc = MouDrawPtr(DeviceHandle);
```

```
HMOU DeviceHandle; /* Mouse device handle */
```

```
USHORT rc; /* return code */
```

## Assembler Language

```
EXTRN MouDrawPtr:FAR
```

```
INCL_MOU EQU 1
```

```
PUSH WORD DeviceHandle ;Mouse device handle
```

```
CALL MouDrawPtr
```

```
Returns WORD
```

This call directs the mouse driver to flush (empty) the mouse event queue and the monitor chain data for the session.

**MouFlushQue (DeviceHandle)**

## Parameters

**DeviceHandle** (*HMOU*) – input

Mouse device handle from a previous MouOpen.

**rc** (*USHORT*) – return

Return code descriptions are:

<b>0</b>	<b>NO_ERROR</b>
<b>385</b>	<b>ERROR_MOUSE_NO_DEVICE</b>
<b>466</b>	<b>ERROR_MOU_DETACHED</b>
<b>501</b>	<b>ERROR_MOUSE_NO_CONSOLE</b>
<b>505</b>	<b>ERROR_MOU_EXTENDED_SG</b>

## C Language

```
#define INCL_MOU
```

```
USHORT rc = MouFlushQue(DeviceHandle);
```

```
HMOU DeviceHandle; /* Mouse device handle */
```

```
USHORT rc; /* return code */
```

## Assembler Language

```
EXTRN MouFlushQue:FAR
```

```
INCL_MOU EQU 1
```

```
PUSH WORD DeviceHandle ;Mouse device handle
```

```
CALL MouFlushQue
```

Returns WORD

---

This call returns status flags for the installed mouse device driver.

<b>MouGetDevStatus</b> ( <b>DeviceStatus</b> , <b>DeviceHandle</b> )
--

### Parameters

**DeviceStatus** (*PUSHORT*) — output

Address of the current status flag settings for the installed mouse device driver.

The return value is a 2-byte set of bit flags.

Bit	Description
15 – 10	Reserved, set to zero.
9	Set if mouse data returned in mickeys, not pels.
8	Set if the drawing operations for pointer draw routine are disabled.
7 – 4	Reserved, set to zero.
3	Set if pointer draw routine disabled by unsupported mode.
2	Set if flush in progress.
1	Set if block read in progress.
0	Set if event queue busy with I/O.

**DeviceHandle** (*HMOU*) — input

Mouse device handle from a previous `MouOpen`.

**rc** (*USHORT*) — return

Return code descriptions are:

0	NO_ERROR
385	ERROR_MOUSE_NO_DEVICE
466	ERROR_MOU_DETACHED
501	ERROR_MOUSE_NO_CONSOLE
505	ERROR_MOU_EXTENDED_SG

### C Language

```
#define INCL_MOU
```

```
USHORT rc = MouGetDevStatus(DeviceStatus, DeviceHandle);
```

```
PUSHORT DeviceStatus; /* Current status flags */  
HMOU DeviceHandle; /* Mouse device handle */
```

```
USHORT rc; /* return code */
```

### Assembler Language

```
EXTRN MouGetDevStatus:FAR  
INCL_MOU EQU 1
```

```
PUSH@ WORD DeviceStatus ;Current status flags  
PUSH WORD DeviceHandle ;Mouse device handle  
CALL MouGetDevStatus
```

Returns WORD

This call returns the current value of the mouse event queue mask.

**MouGetEventMask (EventMask, DeviceHandle)**

## Parameters

**EventMask** (*PUSHORT*) – output

Address in application storage where the current mouse device driver's event mask is returned to the caller by the mouse device driver.

The EventMask is set by MouSetEventMask, and has the following definition:

Bit	Description
15 – 7	Reserved, set to zero.
6	Set to report button 3 press/release events, without mouse motion.
5	Set to report button 3 press/release events, with mouse motion.
4	Set to report button 2 press/release events, without mouse motion.
3	Set to report button 2 press/release events, with mouse motion.
2	Set to report button 1 press/release events, without mouse motion.
1	Set to report button 1 press/release events, with mouse motion.
0	Set to report mouse motion events with no button press/release events.

**DeviceHandle** (*HMOU*) – input

Handle of the mouse device from a previous MouOpen.

**rc** (*USHORT*) – return

Return code descriptions are:

0	NO_ERROR
385	ERROR_MOUSE_NO_DEVICE
466	ERROR_MOU_DETACHED
501	ERROR_MOUSE_NO_CONSOLE
505	ERROR_MOU_EXTENDED_SG

## Remarks

Buttons are logically numbered from left to right.

## C Language

```
#define INCL_MOU
```

```
USHORT rc = MouGetEventMask(EventMask, DeviceHandle);

PUSHORT EventMask; /* Event Mask word */
HMOU DeviceHandle; /* Mouse device handle */

USHORT rc; /* return code */
```

## Assembler Language

```
EXTRN MouGetEventMask:FAR
INCL_MOU EQU 1

PUSH@ WORD EventMask ;Event Mask word
PUSH WORD DeviceHandle ;Mouse device handle
CALL MouGetEventMask
```

Returns WORD

# MouGetNumButtons — Get Number of Mouse Buttons

xPM

This call returns the number of buttons supported on the installed mouse driver.

<b>MouGetNumButtons (NumberOfButtons, DeviceHandle)</b>
---

## Parameters

**NumberOfButtons** (*PUSHORT*) — output

Address of the number of physical buttons. The return values for the number of buttons supported are:

Value	Definition
1	One mouse button
2	Two mouse buttons
3	Three mouse buttons.

**DeviceHandle** (*HMOU*) — input

Handle of the mouse device from a previous MouOpen.

**rc** (*USHORT*) — return

Return code descriptions are:

385	ERROR_MOUSE_NO_DEVICE
466	ERROR_MOU_DETACHED
501	ERROR_MOUSE_NO_CONSOLE
505	ERROR_MOU_EXTENDED_SG

## C Language

```
#define INCL_MOU
```

```
USHORT rc = MouGetNumButtons(NumberOfButtons, DeviceHandle);
```

```
PUSHORT      NumberOfButtons; /* Number of mouse buttons */  
HMOU         DeviceHandle;    /* Mouse device handle */
```

```
USHORT      rc;                /* return code */
```

## Assembler Language

```
EXTRN MouGetNumButtons:FAR  
INCL_MOU EQU 1
```

```
PUSH@ WORD NumberOfButtons ;Number of mouse buttons  
PUSH WORD DeviceHandle ;Mouse device handle  
CALL MouGetNumButtons
```

Returns WORD

This call returns the number of mickeys in each centimeter for the installed mouse driver.

**MouGetNumMickeys** (**NumberOfMickeys**, **DeviceHandle**)

## Parameters

**NumberOfMickeys** (*PUSHORT*) – output

Address of the number of physical mouse motion units. Mouse motion units are reported in mickeys in each centimeter. This value is constant based upon the mouse device attached.

**DeviceHandle** (*HMOU*) – input

Handle of the mouse device from a previous `MouOpen`.

**rc** (*USHORT*) – return

Return code descriptions are:

<b>0</b>	<b>NO_ERROR</b>
<b>385</b>	<b>ERROR_MOUSE_NO_DEVICE</b>
<b>466</b>	<b>ERROR_MOU_DETACHED</b>
<b>501</b>	<b>ERROR_MOUSE_NO_CONSOLE</b>
<b>505</b>	<b>ERROR_MOU_EXTENDED_SG</b>

## C Language

```
#define INCL_MOU
```

```
USHORT rc = MouGetNumMickeys(NumberOfMickeys, DeviceHandle);
```

```
PUSHORT      NumberOfMickeys; /* Number mickeys/centimeter */
HMOU         DeviceHandle;    /* Mouse device handle */
```

```
USHORT      rc;                /* return code */
```

## Assembler Language

```
EXTRN MouGetNumMickeys:FAR
INCL_MOU EQU 1
```

```
PUSH@ WORD    NumberOfMickeys ;Number mickeys/centimeter
PUSH WORD     DeviceHandle    ;Mouse device handle
CALL MouGetNumMickeys
```

Returns WORD



# MouGetNumQueEI — Get Event Queue Status

xPM

This call returns the current status for the mouse device driver event queue.

**MouGetNumQueEI (QueDataRecord, DeviceHandle)**

## Parameters

**QueDataRecord** (*PMOUQUEINFO*) — output

Address of the mouse queue status structure:

**numquelements** (*USHORT*)

Current number of event queue elements, in the range  $0 \leq \text{value} \leq \text{maxnumquelements}$ .

**maxnumquelements** (*USHORT*)

Maximum number of queue elements as specified in the QSIZE = NN parameter in  
DEVICE=MOUSExxx.SYS statement in CONFIG.SYS.

**DeviceHandle** (*HMOU*) — input

Contains the handle of the mouse device obtained from a previous MouOpen.

**rc** (*USHORT*) — return

Return code descriptions are:

0	NO_ERROR
385	ERROR_MOUSE_NO_DEVICE
466	ERROR_MOU_DETACHED
501	ERROR_MOUSE_NO_CONSOLE
505	ERROR_MOU_EXTENDED_SG

## Remarks

The maxnumquelements returned by this function is established during mouse device driver configuration. See the mouse DEVICE=MOUSExxx.SYS statement in the *IBM Operating System/2 Version 1.2 Command Reference* for further details.

## C Language

```
typedef struct _MOUQUEINFO {    /* mouqi */
    USHORT cEvents;             /* current number of event queue elements */
    USHORT cmaxEvents;          /* MaxNumQueElements value */
} MOUQUEINFO;

#define INCL_MOU

USHORT rc = MouGetNumQueEI(QueDataRecord, DeviceHandle);

PMOUQUEINFO    QueDataRecord; /* Ptr to 2-word structure */
HMOU           DeviceHandle;  /* Mouse device handle */

USHORT         rc;            /* return code */
```

**Assembler Language**

```
MOUQUEINFO struc
    mouqi_cEvents    dw ? ;current number of event queue elements
    mouqi_cmaxEvents dw ? ;MaxNumQueueElements value
MOUQUEINFO ends
```

```
EXTRN MouGetNumQueEl:FAR
INCL_MOU      EQU 1
```

```
PUSH@ OTHER  QueDataRecord ;Ptr to 2-word structure
PUSH  WORD   DeviceHandle  ;Mouse device handle
CALL  MouGetNumQueEl
```

Returns WORD

# MouGetPtrPos — Query Mouse Pointer Position

xPM

This call queries the mouse driver to determine the current row and column coordinate position of the mouse pointer.

**MouGetPtrPos (PtrPos, DeviceHandle)**

## Parameters

**PtrPos** (*PPTRLOC*) — output

Address of the mouse pointer position structure:

**pointerrow** (*USHORT*)

Current pointer row coordinate (pels or characters).

**pointercol** (*USHORT*)

Current pointer column coordinate (pels or characters).

**DeviceHandle** (*HMOU*) — input

Contains the handle of the mouse device obtained from a previous `MouOpen`.

**rc** (*USHORT*) — return

Return code descriptions are:

0	NO_ERROR
385	ERROR_MOUSE_NO_DEVICE
466	ERROR_MOU_DETACHED
501	ERROR_MOUSE_NO_CONSOLE
505	ERROR_MOU_EXTENDED_SG

## Remarks

For a text window (VIO) application, the text window is a view on the larger logical video buffer (LVB). The mouse pointer can be outside that view and still be within the extent of the LVB. `MouGetPtrPos` then returns the coordinates of the cell under the mouse pointer. If the pointer is outside the LVB image extent, the coordinates of the nearest LVB cell are returned. In either case, the LVB is scrolled until the reported LVB cell appears within the view window.

## C Language

```
typedef struct _PTRLOC {    /* moupl */
    USHORT row;             /* pointer row coordinate screen
                             position */
    USHORT col;             /* pointer column coordinate screen
                             position */
} PTRLOC;

#define INCL_MOU

USHORT rc = MouGetPtrPos(PtrPos, DeviceHandle);

PPTRLOC    PtrPos;         /* Double word structure */
HMOU       DeviceHandle;    /* Mouse device handle */

USHORT     rc;             /* return code */
```

## Assembler Language

```
PTRLOC struc
    moupl_row dw ? ;pointer row coordinate screen position
    moupl_col dw ? ;pointer column coordinate screen position
PTRLOC ends
```

```
EXTRN MouGetPtrPos:FAR
INCL_MOU EQU 1
```

```
PUSH@ OTHER PtrPos ;Double word structure
PUSH WORD DeviceHandle ;Mouse device handle
CALL MouGetPtrPos
```

Returns WORD

This call allows a process to get (copy) the pointer shape for the session.

**MouGetPtrShape (PtrBuffer, PtrDefRec, DeviceHandle)**

## Parameters

### **PtrBuffer** (*PBYTE*) — output

Address of an area in application storage where the pointer draw device driver returns the pointer bit image. See MouSetPtrShape “MouSetPtrShape — Set Mouse Pointer Shape” on page 4-37 for a further description of the resulting content of this buffer.

### **PtrDefRec** (*PPTRSHAPE*) — input/output

Address of a structure in application storage where the application stores the data necessary for the pointer device driver to return information about the Row by Col image for each bit plane for the mode the display is currently running. See MouSetPtrShape for a further description of the contents of this structure.

### **TotLength** (*USHORT*)

Length of the pointer buffer available for the pointer device driver to build a Row by Col image for each bit plane for the mode the display is currently running. This value is supplied by the application. If the value is too small, pointer draw places the true length of the image in this field, and returns an error.

For all OS/2 system-supported modes, TotLength is specified in bytes and is equal to:

**Mono & Text Modes** For text mode height and width must be 1, so length is always 4.

$$\begin{aligned}\text{TotLength} &= (\text{height in chars}) * (\text{width in chars}) * 2 * 2 \\ &= 1 * 1 * 2 * 2 \\ &= 4\end{aligned}$$

**Graphics Mode** Width-in-pels must be a multiple of 8.

$$\text{TotLength} = (\text{height in pels}) * (\text{width in pels}) * (\text{bits per pel}) * 2 / 8$$

**Modes 4 and 5 (320 X 200)**

$$\text{TotLength} = (\text{height}) * (\text{width}) * 2 * 2 / 8$$

**Mode 6 (640 X 200)**

$$\text{TotLength} = (\text{height}) * (\text{width}) * 1 * 2 / 8$$

Length calculations produce byte boundary buffer sizes.

### **col** (*USHORT*)

Number of columns in the mouse shape. In graphics modes, this field contains the pel width (columns) of the mouse shape for the session and must be greater than or equal to 1. In text modes, col must equal 1.

### **row** (*USHORT*)

Number of rows in the mouse shape. In graphics modes, this field contains the pel height (rows) of the mouse shape for the session and must be greater than or equal to 1. In text modes, row must equal 1.

### **coloffset** (*USHORT*)

This value is returned by the mouse device driver to indicate the relative column offset within the pointer image. The value defines the center (hotspot) of the pointer image. This value is a signed number that represents either character or pel offset, depending on the display mode.

### **rowoffset** (*USHORT*)

This value is returned by the mouse device driver to indicate the relative row offset within the pointer image. The value defines the center (hotspot) of the pointer image. This value is a signed number that represents either character or pel offset, depending on the display mode.

**DeviceHandle** (*HMOU*) — input

Handle of the mouse device from a previous MouOpen.

**rc** (*USHORT*) — return

Return code descriptions are:

<b>0</b>	<b>NO_ERROR</b>
<b>385</b>	<b>ERROR_MOUSE_NO_DEVICE</b>
<b>387</b>	<b>ERROR_MOUSE_INV_PARMS</b>
<b>466</b>	<b>ERROR_MOU_DETACHED</b>
<b>501</b>	<b>ERROR_MOUSE_NO_CONSOLE</b>
<b>505</b>	<b>ERROR_MOU_EXTENDED_SG</b>

## Remarks

The application passes a parameter list with the same meaning as defined for MouSetPtrShape to the mouse device driver. The mouse device driver copies the parameters that describe the pointer shape and attributes into the pointer definition control block pointed to by the PtrDefRec parameter. The word 0 (buffer length = TotLength) pointer definition record parameter field must contain the size in bytes of the application buffer where the device driver is to insert the sessions pointer image. All other words in the parameter list are returned to the application by MouGetPtrShape.

If the buffer size is insufficient, the TotLength field contains the actual size in bytes of the returned pointer image.

The pointer shape may be set by the application with MouSetPtrShape or may be the default image provided by the installed Pointer Device Driver.

## C Language

```
typedef struct _PTRSHAPE { /* moups */
    USHORT cb;             /* total length necessary to build image */
    USHORT col;            /* # of columns in mouse shape */
    USHORT row;            /* number of rows in mouse shape */
    USHORT colHot;         /* column coordinate of pointer image
                           hotspot */
    USHORT rowHot;        /* row coordinate of pointer image
                           hotspot */
} PTRSHAPE;

#define INCL_MOU

USHORT rc = MouGetPtrShape(PtrBuffer, PtrDefRec, DeviceHandle);

PBYTE PtrBuffer; /* Pointer shape buffer */
PPTRSHAPE PtrDefRec; /* Pointer definition struct */
HMOU DeviceHandle; /* Mouse device handle */

USHORT rc; /* return code */
```

# MouGetPtrShape — Get Pointer Shape

xPM

## Assembler Language

```
PTRSHAPE struc
  mouns_cb      dw ? ;total length necessary to build image
  mouns_col     dw ? ;# of columns in mouse shape
  mouns_row     dw ? ;number of rows in mouse shape
  mouns_colHot  dw ? ;column coordinate of pointer image hotspot
  mouns_rowHot  dw ? ;row coordinate of pointer image hotspot
PTRSHAPE ends
```

```
EXTRN  MouGetPtrShape:FAR
INCL_MOU      EQU 1
```

```
PUSH@  OTHER  PtrBuffer      ;Pointer shape buffer
PUSH@  OTHER  PtrDefRec      ;Pointer definition struct
PUSH   WORD   DeviceHandle    ;Mouse device handle
CALL   MouGetPtrShape
```

Returns WORD

This call returns a pair of 1-word scaling factors for the current mouse device.

**MouGetScaleFact (ScaleStruct, DeviceHandle)**

## Parameters

**ScaleStruct (PSCALEFACT)** — output

Address of the control block structure that contains the current row and column coordinate scaling factors. The scaling factors must be greater than or equal to 1 and less than or equal to (32K — 1).

**rowscale (USHORT)**

Row scaling factor.

**colscale (USHORT)**

Column scaling factor.

See "MouSetScaleFact — Set Mouse Scaling Factor" on page 4-40 for more information.

**DeviceHandle (HMOU)** — input

Contains the handle of the mouse device obtained from a previous MouOpen.

**rc (USHORT)** — return

Return code descriptions are:

<b>0</b>	<b>NO_ERROR</b>
<b>385</b>	<b>ERROR_MOUSE_NO_DEVICE</b>
<b>466</b>	<b>ERROR_MOU_DETACHED</b>
<b>501</b>	<b>ERROR_MOUSE_NO_CONSOLE</b>
<b>505</b>	<b>ERROR_MOU_EXTENDED_SG</b>

## Remarks

The units of the scale factor depend on the mode of the display screen for the session. If the screen is operating in text mode, the scaling units are relative to characters. If the screen is operating in graphics mode, the scaling units are relative to pels.

## C Language

```
typedef struct _SCALEFACT {    /* mousc */
    USHORT rowScale;          /* row scaling factor */
    USHORT colScale;          /* column coordinate scaling factor */
} SCALEFACT;

#define INCL_MOU

USHORT rc = MouGetScaleFact(ScaleStruct, DeviceHandle);

PSCALEFACT    ScaleStruct;    /* 2-word structure */
HMOU           DeviceHandle;   /* Mouse device handle */

USHORT         rc;             /* return code */
```



# MouGetScaleFact — Get Mouse Scaling Factors

xPM

## Assembler Language

```
SCALEFACT struc  
    mousc_rowScale dw ? ;row scaling factor  
    mousc_colScale dw ? ;column coordinate scaling factor  
SCALEFACT ends
```

```
EXTRN MouGetScaleFact:FAR  
INCL_MOU EQU 1
```

```
PUSH@ OTHER ScaleStruct ;2-word structure  
PUSH WORD DeviceHandle ;Mouse device handle  
CALL MouGetScaleFact
```

Returns WORD

This call initializes mouse pointer draw support for DOS mode.

<b>MouInitReal (DriverName)</b>
---------------------------------

## Parameters

**DriverName (PSZ)** – input

Address of the name of the Pointer Draw Device Driver used as the pointer-image drawing routine for the DOS mode session.

The name of the device driver must be included in the CONFIG.SYS file at system start-up time.

If the selector portion of the far address is zero and the offset portion is non-zero, the offset portion identifies the power-up display configuration.

**rc (USHORT)** – return

Return code descriptions are:

<b>0</b>	<b>NO_ERROR</b>
<b>385</b>	<b>ERROR_MOUSE_NO_DEVICE</b>
<b>466</b>	<b>ERROR_MOU_DETACHED</b>
<b>412</b>	<b>ERROR_MOUSE_SMG_ONLY</b>
<b>501</b>	<b>ERROR_MOUSE_NO_CONSOLE</b>
<b>505</b>	<b>ERROR_MOU_EXTENDED_SG</b>

## Remarks

MouInitReal is issued by the Base Video Subsystem at system initialization time.

The DOS mode mouse API (INT 33H), in contrast to the OS/2 mode Mouse API, does not contain an OPEN command. In addition, there is only one session for DOS mode.

The default pointer draw routine for DOS mode is located in the same pointer draw device driver, POINTER\$, that is used for OS/2 mode. Establishing addressability to the pointer draw routine must be done during system initialization. This requires passing the entry point of the DOS mode pointer draw routine to the mouse device driver. This is the purpose of the MouInitReal call. It passes the address of the default, power-up pointer draw routine for DOS mode to the mouse device driver. This initialization is transparent to applications.

This call is for use only by the Base Video Subsystem when invoked during system initialization under the shell/session manager PID.

The error code ERROR\_MOUSE\_SMG\_ONLY is valid from shell process only.

## C Language

```
#define INCL_MOU

USHORT rc = MouInitReal(DriverName);

PSZ      DriverName;    /* Pointer draw driver name */

USHORT   rc;            /* return code */
```

# MouInitReal — Initialize DOS mode

xWPM

## Assembler Language

```
EXTRN MouInitReal:FAR
INCL_MOU EQU 1

PUSH@ ASCIIZ DriverName ;Pointer draw driver name
CALL MouInitReal

Returns WORD
```

This call opens the mouse device for the current session.

**MouOpen (DriverName, DeviceHandle)**

## Parameters

**DriverName (PSZ)** – input

DriverName is a far pointer to an ASCIIZ string in application storage containing the name of the pointer draw device driver to be used as the pointer-image drawing routine for this session.

The name of the device driver must be included in the CONFIG.SYS file at system start-up time. Applications that use the default pointer draw device driver supplied by the system must push a double-word of 0s in place of an address.

DriverName has a different definition when the caller is the Base Video Subsystem (BVS). In this case the selector portion of the far address is zero. The offset portion is non-zero and contains a display configuration number (sequentially numbered where 1 is the first display configuration). The MouOpen call issued by BVS is executed on the VioSetMode path. Using the display configuration number passed on the MouOpen call, the Base Mouse Subsystem can detect a change in display configurations. This form of the MouOpen call is not recommended for applications. Applications should either push the far address of an ASCIIZ pointer draw device driver name or push two words of zeros.

**DeviceHandle (PHMOU)** – output

Address of a 1-word value that represents the mouse handle returned to the application.

**rc (USHORT)** – return

Return code descriptions are:

<b>0</b>	<b>NO_ERROR</b>
<b>385</b>	<b>ERROR_MOUSE_NO_DEVICE</b>
<b>390</b>	<b>ERROR_MOUSE_INV_MODULE_PT</b>
<b>466</b>	<b>ERROR_MOU_DETACHED</b>
<b>501</b>	<b>ERROR_MOUSE_NO_CONSOLE</b>
<b>505</b>	<b>ERROR_MOU_EXTENDED_SG</b>

## Remarks

MouOpen initializes the Mouse functions to a known state. The application may have to issue additional mouse functions to establish the environment it desires. For example, after the MouOpen, the collision area is defined to be the size of the entire display. Therefore, to get the pointer to be displayed, the application must issue a MouDrawPtr to remove the collision area.

The state of the mouse after the first MouOpen is:

- Row/Col scale factors set to 16/8. (See "MouSetScaleFact – Set Mouse Scaling Factor" on page 4-40.)
- All events reported. (See "MouSetEventMask – Set Mouse Event Mask" on page 4-33.)
- Empty event queue. (See "MouReadEventQue – Read Mouse Event Queue" on page 4-23 and "MouGetNumQueEl – Get Event Queue Status" on page 4-10.)
- All user settable Device Status bits reset. (Set to zero. See "MouSetDevStatus – Set Mouse Device Status" on page 4-31.)
- Pointer set to center of screen if valid display mode is set. (See "MouSetPtrPos – Set Mouse Pointer Position" on page 4-35.)
- Pointer shape set to the default for the pointer device driver currently registered in the session. (See "MouSetPtrShape – Set Mouse Pointer Shape" on page 4-37.)
- Collision area equal to full screen. (See "MouDrawPtr – Mouse Draw Pointer" on page 4-4 and "MouRemovePtr – Remove Mouse Pointer" on page 4-29.)

# MouOpen — Open Mouse Device

xPM

## C Language

```
#define INCL_MOU

USHORT rc = MouOpen(DriverName, DeviceHandle);

PSZ      DriverName; /* Pointer draw driver name */
PHMOU    DeviceHandle; /* Mouse device handle */

USHORT    rc;          /* return code */
```

## Assembler Language

```
EXTRN MouOpen:FAR
INCL_MOU EQU 1

PUSH@ ASCIIZ DriverName ;Pointer draw driver name
PUSH@ WORD DeviceHandle ;Mouse device handle
CALL MouOpen

Returns WORD
```

This call reads an event from the mouse device FIFO event queue, and places it in a structure provided by the application.

**MouReadEventQue (Buffer, ReadType, DeviceHandle)**

## Parameters

**Buffer** (*PMOUEVENTINFO*) – output

Address of the status of the mouse event queue.

**moustate** (*USHORT*)

State of the mouse at the time of the event.

Bit	Description
15 – 7	Reserved, set to zero.
6	Set if button 3 is down.
5	Set if mouse is moving and button 3 is down.
4	Set if button 2 is down.
3	Set if mouse is moving and button 2 is down.
2	Set if button 1 is down.
1	Set if mouse is moving and button 1 is down.
0	Set if mouse is moving and no buttons are down.

**eventtime** (*ULONG*)

Time stamp (in milliseconds) since the system was started.

**row** (*USHORT*)

Absolute or relative row position.

**col** (*USHORT*)

Absolute or relative column position.

**ReadType** (*PUSHORT*) – input

Address of the action to take when **MouReadEventQue** is issued and the mouse event queue is empty. If the mouse event queue is not empty, this parameter is not examined by the mouse support. **ReadType** values are:

Value	Definition
0	No Wait for data on empty queue (return a NULL record)
1	WAIT for data on empty queue.

**DeviceHandle** (*HMOU*) – input

Handle of the mouse device from a previous **MouOpen**.

**rc** (*USHORT*) – return

Return code descriptions are:

0	NO_ERROR
385	ERROR_MOUSE_NO_DEVICE
387	ERROR_MOUSE_INV_PARAMS
393	ERROR_MOUSE_NO_DATA
466	ERROR_MOU_DETACHED
501	ERROR_MOUSE_NO_CONSOLE
505	ERROR_MOU_EXTENDED_SG

# MouReadEventQuee —

## Read Mouse Event Queue

xPM

### Remarks

The types of queued events are directly affected by the current value of the Mouse EventMask. MouSetEventMask is used to indicate the types of events desired, and MouGetEventMask is used to query the current value of the mask. Refer to these functions for further explanation of the masking of events.

Recognition of the mouse transition depends on the use of MouState returned in the event record. The application should focus on bit transitions that occur in this word. It is important to properly set the event mask with MouSetEventMask for reporting the state transitions.

MouState reports the state of the mouse that resulted from the action that caused the event. The action can be pressing or releasing a button, and/or moving the mouse. All status is given, regardless of the EventMask that was used to determine whether or not to report the event.

For example, assume the EventMask indicates that the application wishes only button 1 events. The EventMask has only bits 1 and 2 set in this case. Also assume the current state of the mouse is no buttons down, and mouse is not moving. At this point, button 1 is pressed causing an event; the status shows button 1 down (bit 2 set). Next the mouse is moved, thereby causing more events; status shows bit 1 set. Finally, mouse is stopped and button 1 is released. The event shows status with no bits set.

Next, button 2 is pressed. No event occurs. Mouse is then moved; again, no event. Then, while mouse is still in motion, button 1 is pressed; an event is generated with bits 1 and 3 set in the state word. While mouse is still in motion, both buttons are released. Because button 1 changes states, an event occurs. The state word has bit 0 set. Finally, mouse is stopped. No event occurs, again because no button 1 transition has taken place.

The Row and Column fields in the Buffer Parameter may contain either absolute display coordinates or relative mouse motion in mickeys. See MouSetDevStatus for additional information.

### C Language

```
typedef struct _MOUEVENTINFO { /* mouev */
    USHORT fs; /* State of mouse at time event was
                reported */
    ULONG time; /* Time since boot in milliseconds */
    USHORT row; /* Absolute/relative row position */
    USHORT col; /* Absolute/relative column position */
}MOUEVENTINFO;

#define INCL_MOU

USHORT rc = MouReadEventQue(Buffer, ReadType, DeviceHandle);

PMQUEVENTINFO Buffer; /* 10 byte Structure address */
PUSHORT ReadType; /* Read type */
HMOU DeviceHandle; /* Mouse device handle */

USHORT rc; /* return code */
```

**Assembler Language**

```
MOUEVENTINFO struc
    mouev_fs dw ? ;State of mouse at time event was reported
    mouev_time dd ? ;time since boot in milliseconds
    mouev_row dw ? ;absolute/relative row position
    mouev_col dw ? ;absolute/relative column position
MOUEVENTINFO ends
```

```
EXTRN MouReadEventQue:FAR
INCL_MOU EQU 1
```

```
PUSH@ OTHER Buffer ;10 byte Structure address
PUSH@ WORD ReadType ;Read type
PUSH WORD DeviceHandle ;Mouse device handle
CALL MouReadEventQue
```

Returns WORD



# MouRegister — Register a Subsystem

xWPM

This call registers a mouse subsystem within a session.

**MouRegister** (*ModuleName*, *EntryName*, *Mask*)

## Parameters

**ModuleName** (*PSZ*) — input

Address of the dynamic link module name. The maximum length is 9 bytes (including ASCIIZ terminator).

**EntryName** (*PSZ*) — input

Address of the dynamic link entry point name of a routine that receives control when any of the registered functions are called. The maximum length is 33 bytes (including ASCIIZ terminator).

**Mask** (*ULONG*) — input

A mask of bits, where each bit set to 1 identifies a mouse function being registered. Bit values are:

Bit	Description
31 – 22	Reserved, set to zero
21	MouSetDevStatus
20	MouFlushQue
19	MouInitReal
18	MouSetPtrPos
17	MouGetPtrPos
16	MouRemovePtr
15	MouDrawPtr
14	MouSetPtrShape
13	MouGetPtrShape
12	MouClose
11	MouOpen
10	Reserved
9	Reserved
8	MouSetEventMask
7	MouSetScaleFact
6	MouGetEventMask
5	MouGetScaleFact
4	MouReadEventQue
3	MouGetNumQueEI
2	MouGetDevStatus
1	MouGetNumMickeyes
0	MouGetNumButtons.

**rc** (*USHORT*) — return

Return code descriptions are:

0	NO_ERROR
385	ERROR_MOUSE_NO_DEVICE
413	ERROR_MOUSE_INVALID_ASCII_Z
414	ERROR_MOUSE_INVALID_MASK
415	ERROR_MOUSE_REGISTER
466	ERROR_MOU_DETACHED
505	ERROR_MOU_EXTENDED_SG

## Remarks

The Base Mouse Subsystem is the default mouse subsystem. There can be only one MouRegister outstanding for each session without an intervening MouDeRegister. MouDeRegister must be issued by the same process that issued MouRegister.

When any registered function is called, control is routed to EntryName. When this routine is entered, four additional values are pushed onto the stack. The first is the index number (Word) of the function being called. The second is a near pointer (Word). The third is the caller's DS register (Word). The fourth is the return address (DWord) to the mouse router. For example, if MouGetNumMickeys were called and control routed to EntryName, the stack would appear as if the following instructions were executed:

```
PUSH WORD    NumberOfMickeys
PUSH WORD    DeviceHandle
CALL FAR     MouGetNumMickeys
PUSH WORD    Function Code
CALL NEAR    Entry point in Mouse Router
PUSH DS
CALL FAR     EntryName.
```

When a registered function returns to the Mouse Router, AX is interpreted as follows:

**AX = 0**

No error. Do not invoke the Base Mouse Subsystem routine. Return AX = 0.

**AX = -1**

Invoke the BaseMouse Subsystem routine. Return AX = return code from the Base Mouse Subsystem.

**AX = error (if not 0 or -1)**

Do not invoke the Base Mouse Subsystem Routine. Return AX = error.

When the mouse router receives a mouse call, it routes it to the Base Mouse Subsystem unless an application or other mouse subsystem has previously issued MouRegister for that call. If the call was registered, the subsystem is entered at the EntryName specified, and provided with the applicable function code.

The registered function mask is used to determine whether a requested function is performed by the registered mouse subsystem or default to the Base Mouse Subsystem.

The following list shows the relationship of the mouse API calls and the Function Code passed to either the Base Mouse Subsystem or a registered mouse subsystem.

<b>MOU API calls</b>	<b>Function Code</b>
MouGetNumButtons	00H
MouGetNumMickeys	01H
MouGetDevStatus	02H
MouGetNumQueEI	03H
MouReadEventQue	03H
MouGetScaleFact	05H
MouGetEventMask	06H
MouSetScaleFact	07H
MouSetEventMask	08H
Reserved	09H
Reserved	0AH
MouOpen	0BH
MouClose	0CH
MouGetPtrShape	0DH
MouSetPtrShape	0EH
MouDrawPtr	0FH
MouRemovePtr	10H
MouGetPtrPos	11H
MouSetPtrPos	12H
MouInitReal	13H

# MouRegister — Register a Subsystem

xWPM

MouFlushQue        14H  
MouSetDevStatus    15H

A registered mouse subsystem must leave the stack, on exit, in the exact state it was received.

## C Language

```
#define INCL_MOU

USHORT rc = MouRegister(ModuleName, EntryName, Mask);

PSZ      ModuleName;    /* Module Name */
PSZ      EntryName;     /* Entry Name */
ULONG    Mask;          /* Function Mask */

USHORT    rc;            /* return code */
```

## Assembler Language

```
EXTRN    MouRegister:FAR
INCL_MOU EQU 1

PUSH@    ASCIIZ    ModuleName    ;Module Name
PUSH@    ASCIIZ    EntryName     ;Entry Name
PUSH     DWORD     Mask          ;Function Mask
CALL     MouRegister

Returns WORD
```

This call allows a process to notify the mouse device driver that the area defined by the passed parameters is for the exclusive use of the application. This area is defined as the *collision* area and is not available to the mouse device driver when drawing pointer images.

**MouRemovePtr (PtrArea, DeviceHandle)**

## Parameters

**PtrArea** (*PNOPTRRECT*) — input

Address of the pointer shape collision area structure:

**leftrow** (*USHORT*)

Upper left row coordinate (pels or characters).

**leftcol** (*USHORT*)

Upper left column coordinate (pels or characters).

**rightrow** (*USHORT*)

Lower right row coordinate (pels or characters).

**rightcol** (*USHORT*)

Lower right column coordinate (pels or characters).

**DeviceHandle** (*HMOU*) — input

Handle of the mouse device from a previous `MouOpen`.

**rc** (*USHORT*) — return

Return code descriptions are:

<b>0</b>	<b>NO_ERROR</b>
<b>385</b>	<b>ERROR_MOUSE_NO_DEVICE</b>
<b>387</b>	<b>ERROR_MOUSE_INV_PARMS</b>
<b>466</b>	<b>ERROR_MOU_DETACHED</b>
<b>501</b>	<b>ERROR_MOUSE_NO_CONSOLE</b>
<b>505</b>	<b>ERROR_MOU_EXTENDED_SG</b>

## Remarks

`MouRemovePtr` may be issued by any process in the session. However, only one collision area is active at a time. Each `MouRemovePtr` command has the effect of resetting the collision area to the location and area specified by the current command.

If the logical pointer position is outside of the collision area specified by the latest `MouRemovePtr` command, the pointer image is drawn.

The `MouDrawPtr` command effectively cancels the `MouRemovePtr` command and allows the pointer to be drawn anywhere on the screen, until a new `MouRemovePtr` command is issued.

# MouRemovePtr — Remove Mouse Pointer

xPM

## C Language

```
typedef struct _NOPTRRECT {    /* mourt */
    USHORT row;               /* upper left row coordinates */
    USHORT col;               /* upper left column coordinates */
    USHORT cRow;
    USHORT cCol;
} NOPTRRECT;

#define INCL_MOU

USHORT rc = MouRemovePtr(PtrArea, DeviceHandle);

PNOPTRRECT    PtrArea;      /* Address of pointer data block */
HMOU          DeviceHandle; /* Mouse device handle */

USHORT        rc;           /* return code */
```

## Assembler Language

```
NOPTRRECT struc
    mourt_row dw ? ;upper left row coordinates
    mourt_col dw ? ;upper left column coordinates
    mourt_cRow dw ?
    mourt_cCol dw ?
NOPTRRECT ends

EXTRN MouRemovePtr:FAR
INCL_MOU EQU 1

PUSH@ OTHER PtrArea      ;Address of pointer data block
PUSH WORD DeviceHandle   ;Mouse device handle
CALL MouRemovePtr

Returns WORD
```

This call sets the mouse device driver status flags for the installed mouse device driver.

**MouSetDevStatus (DeviceStatus, DeviceHandle)**

## Parameters

**DeviceStatus** (*PUSHORT*) – input

Address of the desired status flag settings.

The passed parameter is a 2-byte set of flags. Only the high-order byte has meaning.

Bit	Description
15 – 10	Reserved, set to zero.
9	Set if mouse device is to return data in mickeys.
8	Set if the drawing operations for the pointer draw routine are to be disabled.
7 – 0	Reserved, set to zero.

**DeviceHandle** (*HMOU*) – input

Handle of the mouse device from a previous MouOpen.

**rc** (*USHORT*) – return

Return code descriptions are:

0	NO_ERROR
385	ERROR_MOUSE_NO_DEVICE
387	ERROR_MOUSE_INV_PARAMS
466	ERROR_MOU_DETACHED
501	ERROR_MOUSE_NO_CONSOLE
505	ERROR_MOU_EXTENDED_SG

## Remarks

MouSetDevStatus is the complement to MouGetDevStatus. However, not all status flags may be set with MouSetDevStatus. Only the flags corresponding to the following functions may be modified:

- Return data in mickeys.

Normally, mouse data is returned to the application with the absolute display mode coordinates of the pointer image position on the display screen. By setting this status flag, mouse data is returned in relative mickeys, a unit of mouse movement.

- Don't call pointer draw device.

Normally, the pointer draw device driver is called for all drawing operations. By setting this status flag, the mouse device driver does not call the pointer draw device driver. The application must draw any required pointer image on the screen.

## C Language

```
#define INCL_MOU
```

```
USHORT rc = MouSetDevStatus(DeviceStatus, DeviceHandle);
```

```
PUSHORT DeviceStatus; /* Status flags */
HMOU DeviceHandle; /* Mouse device handle */
```

```
USHORT rc; /* return code */
```

# MouSetDevStatus — Set Mouse Device Status

xPM

## Assembler Language

```
EXTRN  MouSetDevStatus:FAR
INCL_MOU      EQU 1

PUSH@ WORD    DeviceStatus ;Status flags
PUSH  WORD    DeviceHandle ;Mouse device handle
CALL  MouSetDevStatus
```

Returns WORD

This call assigns a new event mask to the current mouse device driver.

**MouSetEventMask (EventMask, DeviceHandle)**

## Parameters

**EventMask** (*PUSHORT*) – input

Address of a value in application storage used to indicate what mouse events are to be placed on the event queue (see *MouReadEventQue*) and which events are to be ignored.

The EventMask bit values are described below:

Bit	Description
15 – 7	Reserved, set to zero.
6	Set to report button 3 press/release events, without mouse motion
5	Set to report button 3 press/release events, with mouse motion
4	Set to report button 2 press/release events, without mouse motion
3	Set to report button 2 press/release events, with mouse motion
2	Set to report button 1 press/release events, without mouse motion
1	Set to report button 1 press/release events, with mouse motion
0	Set to mouse motion events with no button press/release events.

A bit clear setting (set to zero) in an EventMask bit position indicates that the associated type of event is not reported to the application. Note also that the mouse buttons are always numbered from left to right. When the mouse is properly positioned for use, the left-hand button is button 1.

**DeviceHandle** (*HMOU*) – input

Handle of the mouse device from a previous *MouOpen*.

**rc** (*USHORT*) – return

Return code descriptions are:

0	NO_ERROR
385	ERROR_MOUSE_NO_DEVICE
466	ERROR_MOU_DETACHED
501	ERROR_MOUSE_NO_CONSOLE
505	ERROR_MOU_EXTENDED_SG

## Remarks

Setting a bit in the event mask means that the associated event is reported on the mouse FIFO event queue. See “*MouReadEventQue – Read Mouse Event Queue*” on page 4-23 for examples of event mask use.

## C Language

```
#define INCL_MOU
```

```
USHORT rc = MouSetEventMask(EventMask, DeviceHandle);

PUSHORT EventMask; /* Mouse device event mask ptr */
HMOU DeviceHandle; /* Mouse device handle */

USHORT rc; /* return code */
```



# MouSetEventMask — Set Mouse Event Mask

xPM

## Assembler Language

```
EXTRN  MouSetEventMask:FAR
INCL_MOU      EQU 1

PUSH@  WORD    EventMask      ;Mouse device event mask ptr
PUSH   WORD    DeviceHandle    ;Mouse device handle
CALL   MouSetEventMask
```

Returns WORD

---

This call directs the mouse driver to set a new row and column coordinate position for the mouse pointer.

**MouSetPtrPos (PtrPos, DeviceHandle)**

## Parameters

**PtrPos** (*PTRLOC*) – input

Address of the mouse pointer position structure:

**pointerrow** (*USHORT*)

New pointer row coordinate (pels or characters).

**pointercol** (*USHORT*)

New pointer column coordinate (pels or characters).

**DeviceHandle** (*HMOU*) – input

Handle of the mouse device from a previous *MouOpen*.

**rc** (*USHORT*) – return

Return code descriptions are:

<b>0</b>	<b>NO_ERROR</b>
<b>385</b>	<b>ERROR_MOUSE_NO_DEVICE</b>
<b>387</b>	<b>ERROR_MOUSE_INV_PARMS</b>
<b>466</b>	<b>ERROR_MOU_DETACHED</b>
<b>501</b>	<b>ERROR_MOUSE_NO_CONSOLE</b>
<b>505</b>	<b>ERROR_MOU_EXTENDED_SG</b>

## Remarks

The application must ensure that the coordinate position specified conforms to the current display mode orientation for the session. Pel values must be used for graphics modes and character values for text modes.

This function has no effect on the display's current collision area definition as specified by the *MouDrawPtr* call. If the mouse pointer image is directed into a defined collision area, the pointer image is not drawn until either the pointer is moved outside the collision area or the collision area is released by the *MouDrawPtr* call.

## C Language

```
typedef struct _PTRLOC {    /* moupl */
    USHORT row;            /* pointer row coordinate screen
                           position */
    USHORT col;            /* pointer column coordinate screen
                           position */
} PTRLOC;

#define INCL_MOU

USHORT rc = MouSetPtrPos(PtrPos, DeviceHandle);

PTRLOC PtrPos;            /* Double word structure */
HMOU DeviceHandle;        /* Mouse device handle */

USHORT rc;                /* return code */
```

# MouSetPtrPos — Set Mouse Pointer Position

xPM

## Assembler Language

```
PTRLOC struc
    moup1_row dw ? ;pointer row coordinate screen position
    moup1_col dw ? ;pointer column coordinate screen position
PTRLOC ends
```

```
EXTRN MouSetPtrPos:FAR
INCL_MOU EQU 1
```

```
PUSH@ OTHER PtrPos ;Double word structure
PUSH WORD DeviceHandle ;Mouse device handle
CALL MouSetPtrPos
```

Returns WORD

This call allows a process to set the pointer shape and size to be used as the mouse device driver pointer image for all applications in a session.

**MouSetPtrShape (PtrBuffer, PtrDefRec, DeviceHandle)**

## Parameters

### **PtrBuffer (PBYTE) – input**

Address of a buffer containing the bit image used by the mouse device driver as the pointer shape for that session. The buffer consists of AND and XOR pointer masks in a format meaningful to the pointer draw device driver.

For CGA compatible text modes (0, 1, 2, and 3) the following describes the AND and XOR pointer mask bit definitions for each character cell of the masks. Bit values are:

Bit	Description
15	Blinking
14 – 12	Background color
11	Intensity
10 – 8	Foreground color
7 – 0	Character

### **PtrDefRec (PPTRSIZE) – input**

Address of the structure where the application stores the necessary data for the pointer draw device driver to build a row-by-column image for each bit plane for the current display mode. The pointer definition record structure follows:

#### **TotLength (USHORT)**

The total length of the data necessary for the pointer draw device driver to build a row-by-column image for each bit plane for the current display mode.

For all OS/2 system-supported modes, TotLength is specified in bytes and is equal to:

**Mono & Text Modes** For text mode height and width must be 1, so length is always 4.

$$\begin{aligned}\text{TotLength} &= (\text{height in chars}) * (\text{width in chars}) * 2 * 2 \\ &= 1 * 1 * 2 * 2 \\ &= 4\end{aligned}$$

**Graphics Mode** Width-in-pels must be a multiple of 8.

$$\text{TotLength} = (\text{height in pels}) * (\text{width in pels}) * (\text{bits per pel}) * 2 / 8$$

#### **Modes 4 and 5 (320 X 200)**

$$\text{TotLength} = (\text{height}) * (\text{width}) * 2 * 2 / 8$$

#### **Mode 6 (640 X 200)**

$$\text{TotLength} = (\text{height}) * (\text{width}) * 1 * 2 / 8$$

Length calculations produce byte boundary buffer sizes.

#### **col (USHORT)**

Number of columns in the mouse shape. In graphics modes, this field contains the pel width (columns) of the mouse shape for the session and must be greater than or equal to 1. In text modes, col must equal 1.

#### **row (USHORT)**

Number of rows in the mouse shape. In graphics modes, this field contains the pel height (rows) of the mouse shape for the session and must be greater than or equal to 1. In text modes, row must equal 1.

# MouSetPtrShape —

## Set Mouse Pointer Shape

xPM

### **coloffset (USHORT)**

This value is returned by the mouse device driver to indicate the relative column offset within the pointer image. The value defines the center (hotspot) of the pointer image. This value is a signed number that represents either character or pel offset, depending on the display mode.

### **rowoffset (USHORT)**

This value is returned by the mouse device driver to indicate the relative row offset within the pointer image. The value defines the center (hotspot) of the pointer image. This value is a signed number that represents either character or pel offset, depending on the display mode.

### **Programming Note:**

For other custom displays and for the extended modes of the EGA attachment, it is possible to set the display to modes that require multiple bit planes. In these cases, the area sized by the row and column limits must be repeated for each bit plane supported in that mode. Consequently, the calling process must supply enough data to allow the mouse device driver to draw the pointer shape on all currently supported bit planes in that session. For text modes, row and column offset must equal 0.

### **DeviceHandle (HMOU) — input**

Contains the handle of the mouse device obtained from a previous MouOpen.

### **rc (USHORT) — return**

Return code descriptions are:

0	NO_ERROR
385	ERROR_MOUSE_NO_DEVICE
387	ERROR_MOUSE_INV_PARMS
466	ERROR_MOU_DETACHED
501	ERROR_MOUSE_NO_CONSOLE
505	ERROR_MOU_EXTENDED_SG

## **Remarks**

An application passes a data image to the mouse device driver that the mouse driver applies to the screen whenever the logical pointer position is not located in the application-defined collision area. The application synchronizes use of the screen with the mouse driver by way of MouRemovePtr and MouDrawPtr.

The pointer shape is dependent on the display device driver used to support the display device. OS/2 supports text and graphics modes. These modes are restricted to modes 0 through 7, depending on the display device. Character modes (modes 0, 1, 2, 3, and 7) support the pointer cursor only as a reverse block character. This reverse block character has a character height and width equal to 1.

The pointer shape is mapped by the Pointer Draw Device Driver and determined completely by the application. The height and width may vary from 1 through the pel size of the display screen. For restrictions concerning the Pointer Draw Device Driver, see *IBM Operating System/2 Version 1.2 I/O Subsystems And Device Support Volume 1*.

## C Language

```
typedef struct _PTRSHAPE { /* moups */
    USHORT cb; /* total length necessary to build
                image */
    USHORT col; /* # of columns in mouse shape */
    USHORT row; /* number of rows in mouse shape */
    USHORT colHot; /* column coordinate of pointer image
                   hotspot */
    USHORT rowHot; /* row coordinate of pointer image
                   hotspot */
} PTRSHAPE;

#define INCL_MOU

USHORT rc = MouSetPtrShape(PtrBuffer, PtrDefRec, DeviceHandle);

PBYTE      PtrBuffer; /* Pointer shape buffer */
PPTRSHAPE  PtrDefRec; /* Pointer definition record */
HMOU       DeviceHandle; /* Mouse device handle */

USHORT      rc; /* return code */
```

## Assembler Language

```
PTRSHAPE struc
    moups_cb      dw ? ;total length necessary to build image
    moups_col     dw ? ;# of columns in mouse shape
    moups_row     dw ? ;number of rows in mouse shape
    moups_colHot  dw ? ;column coordinate of pointer image hotspot
    moups_rowHot  dw ? ;row coordinate of pointer image hotspot
PTRSHAPE ends

EXTRN MouSetPtrShape:FAR
INCL_MOU      EQU 1

PUSH@ OTHER  PtrBuffer      ;Pointer shape buffer
PUSH@ OTHER  PtrDefRec      ;Pointer definition record
PUSH  WORD   DeviceHandle   ;Mouse device handle
CALL  MouSetPtrShape

Returns WORD
```

# MouSetScaleFact — Set Mouse Scaling Factor

xPM

This call assigns to the current mouse device driver a new pair of 1-word scaling factors.

**MouSetScaleFact** (*ScaleStruct*, *DeviceHandle*)

## Parameters

**ScaleStruct** (*PSCALEFACT*) — input

Address of the control block structure that contains the current row and column coordinate scaling factors. The scaling factors must be greater than or equal to 1 and less than or equal to (32K – 1).

**rowscale** (*USHORT*)

Row scaling factor.

**colscale** (*USHORT*)

Column scaling factor.

**DeviceHandle** (*HMOU*) — input

Handle of the mouse device from a previous *MouOpen*.

**rc** (*USHORT*) — return

Return code descriptions are:

0	NO_ERROR
385	ERROR_MOUSE_NO_DEVICE
387	ERROR_MOUSE_INV_PARMS
466	ERROR_MOU_DETACHED
501	ERROR_MOUSE_NO_CONSOLE
505	ERROR_MOU_EXTENDED_SG

## Remarks

*MouSetScaleFact* sets the mickey-to-pixel ratio for mouse motion. The row scale and column scale ratios specify a number of mickeys for each 8 pixels. The default value for the row scale is 8 mickeys for each 8 pixels. The default value for the column scale is 16 mickeys to 8 pixels.

The number of pixels moved does not have to correspond 1-to-1 with the number of mickeys the mouse moves. The scaling factor defines a sensitivity for the mouse that is a ratio of the number of mickeys required to move the cursor 8 pixels on the screen. The sensitivity determines at what rate the cursor moves on the screen.

## C Language

```
typedef struct _SCALEFACT { /* mousc */
    USHORT rowScale; /* row scaling factor */
    USHORT colScale; /* column coordinate scaling factor */
} SCALEFACT;

#define INCL_MOU

USHORT rc = MouSetScaleFact(ScaleStruct, DeviceHandle);

PSCALEFACT ScaleStruct; /* 2-word structure */
HMOU DeviceHandle; /* Mouse device handle */

USHORT rc; /* return code */
```

## MouSetScaleFact — Set Mouse Scaling Factor

### Assembler Language

```
SCALEFACT struc
    mousc_rowScale dw ? ;row scaling factor
    mousc_colScale dw ? ;column coordinate scaling factor
SCALEFACT ends
```

```
EXTRN MouSetScaleFact:FAR
INCL_MOU EQU 1
```

```
PUSH@ OTHER ScaleStruct ;2-word structure
PUSH WORD DeviceHandle ;Mouse device handle
CALL MouSetScaleFact
```

Returns WORD



# MouSynch —

## Get Synchronous Access

xWPM

---

This call provides synchronous access for a mouse subsystem to the mouse device driver.

<b>MouSynch (IOWait)</b>
--------------------------

### Parameters

**IOWait (USHORT)** — input

Wait for access. The flag Word is defined as follows:

Value	Definition
0	Control immediately returned to requestor.
1	Requestor waits until mouse device driver is free.

**rc (USHORT)** — return

Return code descriptions are:

0	NO_ERROR
121	ERROR_SEM_TIMEOUT

### Remarks

MouSynch blocks all other threads within a session until the semaphore clears (returns from the subsystem to the router). To ensure proper synchronization, MouSynch should be issued by a mouse subsystem if it intends to access dynamically modifiable shared data for each session or if it intends to issue a DosDevIOctl. MouSynch does not protect globally shared data from threads in other sessions.

### C Language

```
#define INCL_MOUSE

USHORT rc = MouSynch(IOWait);

USHORT      IOWait;      /* Indicate wait/no wait */
USHORT      rc;          /* return code */
```

### Assembler Language

```
EXTRN MouSynch:FAR
INCL_MOUSE EQU 1

PUSH WORD IOWait ;Indicate wait/no wait
CALL MouSynch

Returns WORD
```

---

## Chapter 5. Video Function Calls

This chapter reflects the Video API interface of OS/2 only.

If `ERROR_VIO_SEE_ERROR_LOG` is returned, further information about the error that occurred can be obtained by calling `WinGetLast Error`.

### Notes:

1. Calls marked xPM are not supported by Presentation Manager, and must not be used by Presentation Manager applications. An error code is returned if any of these calls are issued.
2. Calls marked xWPM are not windowable and are not supported by Presentation Manager. They can be used in OS/2 mode.
3. Calls marked FAPI are present in the Family API.
4. Those calls without an icon are:
  - Windowable
  - Presentation Manager
  - Full-screen
  - Not Family API.

# VioDeRegister — DeRegister Video Subsystem

xWPM

---

This call deregisters a video subsystem previously registered within a session.

<b>VioDeRegister ( )</b>
--------------------------

## Parameters

**rc** (*USHORT*) — return

Return code descriptions are:

<b>0</b>	<b>NO_ERROR</b>
<b>404</b>	<b>ERROR_VIO_DEREGISTER</b>
<b>430</b>	<b>ERROR_VIO_ILLEGAL_DURING_POPUP</b>
<b>465</b>	<b>ERROR_VIO_DETACHED</b>
<b>494</b>	<b>ERROR_VIO_EXTENDED_SG</b>

## Remarks

VioDeRegister must be issued by the same process that issued the previous VioRegister. After VioDeRegister is issued, subsequent video calls are processed by the Base Video Subsystem.

## C Language

```
#define INCL_VIO
```

```
USHORT rc = VioDeRegister(VOID);
```

```
USHORT rc; /* return code */
```

## Assembler Language

```
EXTRN VioDeRegister:FAR  
INCL_VIO EQU 1
```

```
CALL VioDeRegister
```

```
Returns WORD
```

# VioEndPopUp – Deallocate Pop-up Display Screen

This call is issued by the application when it no longer requires the temporary screen obtained through a previous VioPopUp call.

**VioEndPopUp (VioHandle)**

## Parameters

**VioHandle (HVIO)** – input  
A reserved word of 0s.

**rc (USHORT)** – return  
Return code descriptions are:

0	NO_ERROR
405	ERROR_VIO_NO_POPUP
436	ERROR_VIO_INVALID_HANDLE

## Remarks

When the application issues a VioEndPopUp call, all video calls are directed to the application's normal video buffer.

## PM Considerations

An error is returned if issued with a non-zero handle.

## C Language

```
#define INCL_VIO

USHORT rc = VioEndPopUp(VioHandle);

HVIO          VioHandle;    /* Vio device handle */

USHORT        rc;           /* return code */
```

## Assembler Language

```
EXTRN VioEndPopUp:FAR
INCL_VIO EQU 1

PUSH WORD VioHandle ;Vio device handle
CALL VioEndPopUp

Returns WORD
```

# VioGetAnsi – Get ANSI Status

---

This call returns the current ANSI status On/Off state.

<b>VioGetAnsi</b> ( <i>Indicator</i> , <i>VioHandle</i> )
---

## Parameters

**Indicator** (*PUSHORT*) – output

Address of the current ANSI status. A value of 1 indicates ANSI is active, and a value of 0 indicates ANSI is not active.

**VioHandle** (*HVIO*) – input

This must be zero unless the caller is a Presentation Manager application, in which case it must be the value returned by VioGetPs.

**rc** (*USHORT*) – return

Return code descriptions are:

<b>0</b>	<b>NO_ERROR</b>
<b>436</b>	<b>ERROR_VIO_INVALID_HANDLE</b>
<b>465</b>	<b>ERROR_VIO_DETACHED</b>

## C Language

```
#define INCL_VIO
```

```
USHORT rc = VioGetAnsi(Indicator, VioHandle);
```

```
PUSHORT Indicator; /* On/Off indicator (returned) */
HVIO VioHandle; /* Vio handle */

USHORT rc; /* return code */
```

## Assembler Language

```
EXTRN VioGetAnsi:FAR
INCL_VIO EQU 1

PUSH@ WORD Indicator ;On/Off indicator (returned)
PUSH WORD VioHandle ;Vio handle
CALL VioGetAnsi
```

Returns WORD

# VioGetBuf – Get Logical Video Buffer

This call returns the address of the logical video buffer (LVB).

**VioGetBuf (LVBPtr, Length, VioHandle)**

## Parameters

**LVBPtr** (*PULONG*) – output

Address of the selector and offset of the logical video buffer. Applications should not assume the offset portion of this far address is 0.

**Length** (*PUSHORT*) – output

Address of the length buffer in bytes. The length is: number of rows \* number of columns \* size of cell.

**VioHandle** (*HVIO*) – input

This must be zero unless the caller is a Presentation Manager application, in which case it must be the value returned by VioGetPs.

**rc** (*USHORT*) – return

Return code descriptions are:

<b>0</b>	<b>NO_ERROR</b>
<b>355</b>	<b>ERROR_VIO_MODE</b>
<b>430</b>	<b>ERROR_VIO_ILLEGAL_DURING_POPUP</b>
<b>436</b>	<b>ERROR_VIO_INVALID_HANDLE</b>
<b>465</b>	<b>ERROR_VIO_DETACHED</b>

## Remarks

An application using VioGetBuf can prepare a screen in the application's own logical video buffer (LVB) offline. When the application is in the foreground, the physical screen buffer is updated from the LVB when VioShowBuf is issued. When the application runs in the background, the physical screen buffer is updated when the application is switched to the foreground.

Once VioGetBuf is issued, all VioWrtXX calls issued while the application is running in the foreground are written to the physical display buffer and LVB. If a VioGetPhysBuf is subsequently issued, then the VioWrtXX calls are only written to the physical display buffer. They are no longer written to the LVB.

VioGetMode may be used to determine the dimensions of the buffer.

If VioSetMode is issued following a VioGetBuf call, the size of the logical video buffer is adjusted to correspond to the new mode. There is one logical video buffer per session (or presentation space if AVIO application) that corresponds to the current mode on the current display configuration.

## PM Considerations

This function returns the address and length of the Advanced VIO presentation space. The presentation space may be used to directly manipulate displayed information.

## C Language

```
#define INCL_VIO
```

```
USHORT rc = VioGetBuf(LVBPtr, Length, VioHandle);
```

```
PULONG      LVBPtr;      /* Points to LVB */
PUSHORT     Length;      /* Length of buffer */
HVIO        VioHandle;    /* Vio handle */

USHORT      rc;           /* return code */
```

# VioGetBuf — Get Logical Video Buffer

## Assembler Language

```
EXTRN VioGetBuf:FAR
INCL_VIO      EQU 1

PUSH@ DWORD  LVBPtr      ;Points to LVB
PUSH@ WORD   Length      ;Length of buffer
PUSH  WORD   VioHandle    ;Vio handle
CALL  VioGetBuf

Returns WORD
```

This call returns the video display configuration.

<b>VioGetConfig (ConfigID, ConfigData, VioHandle)</b>
---

## Parameters

**ConfigID (USHORT)** – input

Identifies for which display configuration information is being requested:

Value	Definition
0	Current configuration
1	Primary configuration
2	Secondary configuration.

For OS/2 1.2, when ConfigID = 0, the current configuration is returned rather than the primary configuration (as was returned in OS/2 1.0 and 1.1). This change makes the OS/2 mode version of VioGetConfig match the family API version that has returned the current configuration starting with OS/2 1.0. OS/2 1.0 and 1.1 applications that issued VioGetConfig to determine the display configuration benefit from this change. The application can run on the configuration selected by the operator (by issuing the MODE command before invoking the application) rather than switching away from the operator selected display.

**ConfigData (PVIOCONFIGINFO)** – output

Address of structure where the display configuration is returned.

**length (USHORT)**

Input parameter to VioGetConfig. Length specifies the length of the data structure in bytes including Length itself. The maximum size structure required in OS/2 1.0 and 1.1 is 10 bytes.

The maximum size structure required in OS/2 1.2 is variable and can be determined by issuing VioGetConfig with Length set to 2. When Length is set to 2 on input, VioGetConfig returns the size of the maximum structure required in the Length field on output. When Length is not equal to 2 on input, the Length field is modified on output to reflect the actual number of bytes returned. That is, if more than the maximum size was specified, the maximum size is returned. However, if less than the maximum size is specified, the value returned reflects the number of bytes of complete fields returned.

**adapertype (USHORT)**

Display adapter type.

Value	Definition
0	Monochrome-compatible
1	Color Graphics Adapter (CGA)
2	Enhanced Graphics Adapter (EGA)
3	VGA or PS/2 Display Adapter
4 – 6	Reserved
7	IBM Personal System/2 Display Adapter 8514/A.

Values ranging from 0 – 4095 are reserved for IBM.

**displaytype (USHORT)**

Display or monitor type.

Value	Definition
0	Monochrome display
1	Color display
2	Enhanced Color Display
3	PS/2 Monochrome Display 8503
4	PS/2 Color Displays 8512 and 8513
5 – 8	Reserved
9	PS/2 Color Display 8514



# VioGetConfig —

## Get Video Configuration

FAP1

10	IBM Plasma Display Panel
11	Monochrome Displays 8507 and 8604
12	Reserved

Values ranging from 0 – 4095 are reserved for IBM.

### **adaptmem (ULONG)**

Amount of memory, in bytes, on the adapter.

### **Configuration # (USHORT)**

Number of the display configuration that this data corresponds to. This is assigned by the video subsystem, not the Base Video Handler (BVH).

### **VDHVersion (USHORT)**

This field is reserved.

### **Flag bits (USHORT)**

Are defined as follows:

Bit	Description
15 – 1	Reserved
0	Power up display configuration.

### **Hardware state buffer size (ULONG)**

Size of the buffer required by the Base Video Handler (BVH) to save the full hardware state excluding the physical display buffer.

### **Max buffer size – full save (ULONG)**

Maximum size buffer required by the BVH to save the full physical display buffer.

### **Max buffer size – partial save (ULONG)**

Maximum size buffer required by the BVH to save the portion of the physical display buffer that is overlaid by a pop-up.

### **Offset to emulated adapter types (USHORT)**

Offset within the configuration data structure to the following information describing what other display adapters are emulated by this display adapter.

### **Number of Data words (USHORT)**

Contains a one word field specifying a count of data words to follow.

### **Data word 1 (USHORT)**

Bits set in the data words identify display adapters emulated. Data word 1 has the following definition:

Bit	Description
0	Monochrome/printer adapter
1	Color graphics adapter
2	Enhanced graphics adapter
3	VGA or PS/2 display adapter
4 – 6	Reserved
7	8514/A Adapter
8 – 15	Reserved.

### **Data word 2 (USHORT)**

Reserved.

### **Data word N (USHORT)**

Reserved.

### **Offset to emulated display types (USHORT)**

Offset within the configuration data structure to the following information describing what other displays are emulated by this display.

### **Number of Data words (USHORT)**

One word field specifying a count of data words to follow.

## VioGetConfig — Get Video Configuration

### Data word 1 (*USHORT*)

Bits set in the data words identify displays emulated. Data word 1 has the following definition:

Bit	Description
0	5151 monochrome display
1	5153 color display
2	5154 enhanced color display
3	8503 monochrome display
4	8512 or 8513 color display
5–8	Reserved
9	8514 color display
10	IBM Plasma Display Panel
11	Monochrome Displays 8507 and 8604
12–15	Reserved.

### Data word 2 (*USHORT*)

Reserved

### Data word N (*USHORT*)

Reserved.

### VioHandle (*HVIO*) — input

This must be zero unless the caller is a Presentation Manager application, in which case it must be the value returned by VioGetPs.

### rc (*USHORT*) — return

Return code descriptions are:

0	NO_ERROR
421	ERROR_VIO_INVALID_PARMS
436	ERROR_VIO_INVALID_HANDLE
438	ERROR_VIO_INVALID_LENGTH
465	ERROR_VIO_DETACHED

### Remarks

The values returned may not be correct if the adapter cannot be properly identified by the Base Video Handler (BVH) selected at system installation time. It can also be incorrect if the physical setup does not match that indicated by the presence of the adapter or by adapter switches. For example, it is impossible to detect the absence of a display on a CGA or the display attached to an EGA, despite the setup switches.

# VioGetConfig — Get Video Configuration

FAP1

## C Language

```
typedef struct _VIOCONFIGINFO { /* vioin */
    USHORT cb; /* Length of this data structure */
    USHORT adapter; /* Display adapter type */
    USHORT display; /* Display/monitor type */
    ULONG cbMemory; /* Amount of memory on the adapter in bytes */
    USHORT Configuration;
    USHORT VDHVersion;
    USHORT Flags;
    ULONG HWBufferSize;
    ULONG FullSaveSize;
    ULONG PartSaveSize;
    USHORT EMAdaptersOFF; /* Offset to emulated adapter types */
    USHORT EMDisplaysOFF; /* Offset to emulated display types */
} VIOCONFIGINFO;

#define INCL_VIO

USHORT rc = VioGetConfig(ConfigID, ConfigData, VioHandle);

USHORT ConfigID; /* Configuration ID */
PVIOCONFIGINFO ConfigData; /* Configuration data */
HVIO VioHandle; /* Vio handle */

USHORT rc; /* return code */
```

## Assembler Language

```
VIOCONFIGINFO struc
    vioin_cb dw ? ;Length of this data structure
    vioin_adapter dw ? ;Display adapter type
    vioin_display dw ? ;Display/monitor type
    vioin_cbMemory dd ? ;Amount of memory on the adapter in bytes
    vioin_Configuration dw ? ;
    vioin_VDHVersion dw ? ;
    vioin_Flags dw ? ;
    vioin_HWBufferSize dd ? ;
    vioin_FullSaveSize dd ? ;
    vioin_PartSaveSize dd ? ;
    vioin_EMAdaptersOFF dw ? ;Offset to emulated adapter types
    vioin_EMDisplaysOFF dw ? ;Offset to emulated display types
VIOCONFIGINFO ends

EXTRN VioGetConfig:FAR
INCL_VIO EQU 1

PUSH WORD ConfigID ;Configuration ID
PUSH@ OTHER ConfigData ;Configuration data
PUSH WORD VioHandle ;Vio handle
CALL VioGetConfig
```

Returns WORD

# VioGetCp – Query Video Code Page

This call allows a process to query the code page currently used to display text data.

**VioGetCp (Reserved, CodePageID, VioHandle)**

## Parameters

**Reserved** (*USHORT*) – input  
A reserved word of 0s.

**CodePageID** (*PUSHORT*) – output  
Address of a word in the application's data area. The current video code page is returned in this word.

**VioHandle** (*HVIO*) – input  
This must be zero unless the caller is a Presentation Manager application, in which case it must be the value returned by VioGetPs.

**rc** (*USHORT*) – return  
Return code descriptions are:

0	NO_ERROR
355	ERROR_VIO_MODE
436	ERROR_VIO_INVALID_HANDLE
465	ERROR_VIO_DETACHED
468	ERROR_VIO_USER_FONT

## Remarks

The display code page ID previously set by VioSetCp, or inherited from the requesting process, is returned to the caller.

The code page tag returned is the currently active code page. A value of 0000 indicates that the code page in use is the ROM code page provided by the hardware.

If ERROR\_VIO\_USER\_FONT is returned, it indicates a user font that was previously loaded with VioSetFont is the active code page.

## C Language

```
#define INCL_VIO

USHORT rc = VioGetCp(Reserved, CodePageID, VioHandle);

USHORT      Reserved;      /* Reserved (must be zero) */
PUSHORT     CodePageID;    /* Code page ID */
HVIO        VioHandle;     /* Video handle */

USHORT      rc;            /* return code */
```

## Assembler Language

```
EXTRN VioGetCp:FAR
INCL_VIO EQU 1

PUSH WORD Reserved ;Reserved (must be zero)
PUSH@ WORD CodePageID ;Code page ID
PUSH WORD VioHandle ;Video handle
CALL VioGetCp

Returns WORD
```

---

This call returns the coordinates of the cursor.

<b>VioGetCurPos</b> (Row, Column, VioHandle)
--

## Parameters

**Row** (*PUSHORT*) — output

Address of the current Row position of the cursor where 0 is the top row.

**Column** (*PUSHORT*) — output

Address of the current column position of the cursor where 0 is the leftmost column.

**VioHandle** (*HVIO*) — input

This must be zero unless the caller is a Presentation Manager application, in which case it must be the value returned by VioGetPs.

**rc** (*USHORT*) — return

Return code descriptions are:

0	NO_ERROR
355	ERROR_VIO_MODE
436	ERROR_VIO_INVALID_HANDLE
465	ERROR_VIO_DETACHED

## C Language

```
#define INCL_VIO
```

```
USHORT rc = VioGetCurPos(Row, Column, VioHandle);
```

```
PUSHORT    Row;           /* Row return data */
PUSHORT    Column;        /* Column return data */
HVIO       VioHandle;      /* Vio handle */
```

```
USHORT     rc;             /* return code */
```

## Assembler Language

```
EXTRN VioGetCurPos:FAR
INCL_VIO EQU 1
```

```
PUSH@ WORD Row           ;Row return data
PUSH@ WORD Column        ;Column return data
PUSH WORD VioHandle       ;Vio handle
CALL VioGetCurPos
```

Returns WORD

This call returns the cursor type.

**VioGetCurType (CursorData, VioHandle)**

## Parameters

**CursorData** (*PVIOCURSORINFO*) – output

Address of the cursor characteristics structure:

**startline** (*USHORT*)

Horizontal scan line in the character cell that marks the top line of the cursor. If the character cell has *n* scan lines, 0 is the top scan line of the character cell and (*n*–1) is the bottom scan line.

**endline** (*USHORT*)

Horizontal scan line in the character cell that marks the bottom line of the cursor. Scan lines within a character cell are numbered as defined in startline.

**cursorwidth** (*USHORT*)

Width of the cursor. In text modes, cursorwidth is the number of columns. The maximum number supported by the OS/2 base video subsystem is 1. In graphics modes, cursorwidth is the number of pels.

**cursorattrib** (*USHORT*)

A value of –1 denotes a hidden cursor, all other values in text mode denote normal cursor and in graphics mode denote color attribute.

**VioHandle** (*HVIO*) – input

This must be zero unless the caller is a Presentation Manager application, in which case it must be the value returned by VioGetPs.

**rc** (*USHORT*) – return

Return code descriptions are:

<b>0</b>	<b>NO_ERROR</b>
<b>355</b>	<b>ERROR_VIO_MODE</b>
<b>436</b>	<b>ERROR_VIO_INVALID_HANDLE</b>
<b>465</b>	<b>ERROR_VIO_DETACHED</b>

## Remarks

If CursorStartLine and CursorEndLine were originally specified as percentages on VioSetCurType (using negative values), the positive values into which they were translated are returned. Refer to VioSetCurType for more information on how percentages can be used to set CursorStartLine and CursorEndLine independent of the number of scan lines per character cell.

## Family API Considerations

In DOS mode, VioGetCurType returns only two values for cursorattrib: 0 = visible cursor, and –1 = hidden cursor.

# VioGetCurType – Get Cursor Type

FAPi

## C Language

```
typedef struct _VIOCURSORINFO { /* vioci */
    USHORT yStart; /*cursor start line */
    USHORT cEnd; /* cursor end line */
    USHORT cx; /* cursor width */
    USHORT attr; /* -1=hidden cursor, any other=normal
                  cursor */
} VIOCURSORINFO;

#define INCL_VIO

USHORT rc = VioGetCurType(CursorData, VioHandle);

PVIOCURSORINFO CursorData; /* Cursor characteristics */
HVIO VioHandle; /* Vio handle */

USHORT rc; /* return code */
```

## Assembler Language

```
VIOCURSORINFO struc
    vioci_yStart dw ? ;cursor start line
    vioci_cEnd dw ? ;cursor end line
    vioci_cx dw ? ;cursor width
    vioci_attr dw ? ;-1=hidden cursor, any other=normal cursor
VIOCURSORINFO ends
```

```
EXTRN VioGetCurType:FAR
INCL_VIO EQU 1
```

```
PUSH@ OTHER CursorData ;Cursor characteristics
PUSH WORD VioHandle ;Vio handle
CALL VioGetCurType
```

Returns WORD

---

This call returns either the font table of the size specified or the font in use.

**VioGetFont (RequestBlock, VioHandle)**

## Parameters

**RequestBlock** (*PVIOFONTINFO*) – input/output

Address of the font structure that returns current RAM font or specified ROM or code page font depending on the request type:

**length** (*USHORT*)

Length of structure, including length.

**14** Only valid value.

**reqtype** (*USHORT*)

Request type:

Value	Definition
0	Get current RAM font for EGA, VGA, or IBM Personal System/2 Display Adapter.
1	Get ROM font for CGA, EGA, VGA, or IBM Personal System/2 Display Adapter.

**pelcolumns** (*USHORT*)

Pel columns in character cell.

**pelrows** (*USHORT*)

Pel rows in character cell.

**fonttable** (*PVOID*)

Address of the requested font table returned in a caller-supplied data area. If the storage area is accessed by way of an address of 0, a system-supplied segment containing the requested font table is returned.

**tablelength** (*USHORT*)

Length, in bytes, of the caller-supplied data area where the font table is returned.

**VioHandle** (*HVIO*) – input

Reserved word of 0s.

**rc** (*USHORT*) – return

Return code descriptions are:

0	NO_ERROR
355	ERROR_VIO_MODE
421	ERROR_VIO_INVALID_PARMS
438	ERROR_VIO_INVALID_LENGTH
465	ERROR_VIO_DETACHED
467	ERROR_VIO_FONT
494	ERROR_VIO_EXTENDED_SG

## Remarks

For reqtype = 1, return ROM font, the font size requested must be supported by the display adapter installed. The 8x8, 8x14, 9x14, 8x16, or 9x16 character font may be requested for the VGA or PS/2 Display Adapters. The 8x8, 8x14, or 9x14 font may be requested for the enhanced graphics adapter. The 8x8 font may be requested for the color graphics adapter.

**Note:** Although graphics mode support is provided in VioGetFont, this support is not provided by the Base Video Handlers provided with OS/2.

For reqtype = 1, return ROM font, the far address returned is a ROM pointer only for those fonts where the font table for the full 256-character set is actually contained in ROM. Otherwise, the far address returned is a RAM pointer. Note that for 8x8 on the CGA, the font table for the full 256-character set is



returned. For 9x14 or 9x16 the font table for the full 256-character set is also returned. Partial fonts are not returned. The 9x14 and 9x16 fonts are derived from variations of the 8x14 and 8x16 fonts, respectively, where the definitions of fonts for those characters that are different, are replaced.

For VioGetFont specifying reqtype = 1, return ROM font, the font returned is derived from the fonts contained in the system, EGA, VGA, and PS/2 Display Adapter BIOS data areas as applicable. There is an exception for the EGA, VGA and PS/2 Display Adapter when VioSetCp or VioSetFont has been issued. In that case, the font of the size requested is returned from the active code page or the list of user fonts already set.

### C Language

```
typedef struct _VIOFONTINFO { /* viofi */
    USHORT cb; /* length of this structure */
    USHORT type; /* request type */
    USHORT cxCell; /* pel columns in character cell */
    USHORT cyCell; /* pel rows in character cell */
    PVOID pbData; /* requested font table (returned) */
    USHORT cbData; /* length of caller supplied data area
                    (in bytes) */
} VIOFONTINFO;

#define INCL_VIO

USHORT rc = VioGetFont(RequestBlock, VioHandle);

PVIOFONTINFO RequestBlock; /* Request block */
HVIO VioHandle; /* Vio handle */

USHORT rc; /* return code */
```

### Assembler Language

```
VIOFONTINFO struc
    viofi_cb dw ? ;length of this structure
    viofi_type dw ? ;request type
    viofi_cxCell dw ? ;pel columns in character cell
    viofi_cyCell dw ? ;pel rows in character cell
    viofi_pbData dd ? ;requested font table (returned)
    viofi_cbData dw ? ;length of caller supplied data area (in bytes)
VIOFONTINFO ends

EXTRN VioGetFont:FAR
INCL_VIO EQU 1

PUSH@ OTHER RequestBlock ;Request block
PUSH WORD VioHandle ;Vio handle
CALL VioGetFont

Returns WORD
```

---

This call returns the mode of the display.

<b>VioGetMode (ModeData, VioHandle)</b>
---

## Parameters

**ModeData** (*PVIOMODEINFO*) – input/output

Far address of a structure where mode characteristics are returned.

**length** (*USHORT*)

Input parameter to VioGetMode. Length specifies the length of the data structure in bytes including Length itself. The value specified on input controls the amount of mode data returned. The minimum structure size required is 2 bytes, and the maximum structure size required is 34 bytes. For OS/2 1.2, a length of 2 returns the size of the maximum structure required for all the mode data. When length is not equal to 2, the length field is modified on output to reflect the actual number of bytes returned.

**type** (*UCHAR*)

Mode characteristics bit mask:

Bit	Description
7 – 4	Reserved
3	0 = VGA-compatible modes 0 thru 13H 1 = Native mode
2	0 = Enable color burst 1 = Disable color burst
1	0 = Text mode 1 = Graphics mode
0	0 = Monochrome compatible mode 1 = Other.

**numcolors** (*UCHAR*)

Number of colors defined as a power of 2. This is equivalent to the number of color bits that define the color, for example:

Value	Definition
0	Monochrome modes 7, 7+, and F.
1	2 colors
2	4 colors
4	16 colors
8	256 colors

**textcols** (*USHORT*)

Number of text columns.

**textrows** (*USHORT*)

Number of text rows.

**pelcols** (*USHORT*)

Horizontal resolution, number of pel columns.

**pelrows** (*USHORT*)

Vertical resolution, number of pel rows.

**Attribute Format** (*UCHAR*)

Format of the attributes.

# VioGetMode — Get Display Mode

FAPI xPM

## Number of Attributes (*UCHAR*)

Number of attributes in a character cell.

## Buffer Address (*ULONG*)

32-bit physical address of the physical display buffer for this mode.

## Buffer Length (*ULONG*)

Length of the physical display buffer for this mode.

## Full Buffer Size (*ULONG*)

Size of the buffer required for a full save of the physical display buffer for this mode.

## Partial Buffer Size (*ULONG*)

Size of the buffer required for a partial (pop-up) save of the physical display buffer for this mode.

## Extended Data Area Address (*PCH*)

Far address to an extended mode data structure or zero if none. The format of the extended mode data structure is determined by the device driver and is unknown to OS/2.

## VioHandle (*HVIO*) — input

Reserved word of 0s.

## rc (*USHORT*) — return

Return code descriptions are:

0	NO_ERROR
436	ERROR_VIO_INVALID_HANDLE
438	ERROR_VIO_INVALID_LENGTH
465	ERROR_VIO_DETACHED
494	ERROR_VIO_EXTENDED_SG

## Remarks

Refer to "VioSetMode — Set Display Mode" on page 5-69 for examples.

## C Language

```
typedef struct _VIOMODEINFO {
    USHORT cb; /* Length of the entire data structure */
    UCHAR fbType; /* Bit mask of mode being set */
    UCHAR color; /* Number of colors (power of 2) */
    USHORT col; /* Number of text columns */
    USHORT row; /* Number of text rows */
    USHORT hres; /* Horizontal resolution */
    USHORT vres; /* Vertical resolution */
    UCHAR fmt_ID; /* Attribute format */
    UCHAR attrib; /* Number of attributes */
    ULONG buf_addr;
    ULONG buf_length;
    ULONG full_length;
    ULONG partial_length;
    PCH ext_data_addr;
} VIOMODEINFO;
typedef VIOMODEINFO far *PVIOMODEINFO;

#define INCL_VIO

USHORT rc = VioGetMode(ModeData, VioHandle);

PVIOMODEINFO ModeData; /* Mode characteristics */
HVIO VioHandle; /* Vio handle */

USHORT rc; /* return code */
```

## Assembler Language

```

VIOMODEINFO struc
    viomi_cb          dw ? ;Length of the entire data structure
    viomi_fbType      db ? ;Bit mask of mode being set
    viomi_color       db ? ;Number of colors (power of 2)
    viomi_col         dw ? ;Number of text columns
    viomi_row         dw ? ;Number of text rows
    viomi_hres        dw ? ;Horizontal resolution
    viomi_vres        dw ? ;Vertical resolution
    viomi_fmt_ID      db ? ;Attribute format
    viomi_attrib       db ? ;Number of attributes
    viomi_buf_addr    dd ? ;
    viomi_buf_length  dd ? ;
    viomi_full_length dd ? ;
    viomi_partial_length dd ? ;
    viomi_ext_data_addr dd ? ;
VIOMODEINFO ends

```

```

EXTRN VioGetMode:FAR
INCL_VIO      EQU 1

```

```

PUSH@ OTHER ModeData      ;Mode characteristics
PUSH WORD VioHandle       ;Vio handle
CALL VioGetMode

```

Returns WORD

This call gets addressability to the physical display buffer.

**VioGetPhysBuf (DisplayBuf, Reserved)**

## Parameters

**DisplayBuf(PVIOPHYSBUF)** – input/output

Address of the data structure that contains the physical display buffer address and length on input and returns the selectors used to address the display buffer.

**displaybufaddr (PBYTE)**

Address of the 32 bit start address (selector:offset) of the physical display buffer passed as input. If displaybuflen is 0, then displaybufaddr is the far address of the PhysBuf Block described below.

**displaybuflen (ULONG)**

32 bit length of the physical display buffer. If displaybuflen is 0, then displaybufaddr is treated as the far address of the PhysBuf Block described below and the Selector List is not present.

**selectors (SEL)**

Selector list.

Returns the selectors (each of word-length) that address the physical display buffer. The first selector returned in the list, addresses the first 64KB of the physical display buffer or displaybuflen, whichever is smaller. If displaybuflen is greater than 64KB, the second selector addresses the second 64KB.

The last selector returned in the list, addresses the remainder of the display buffer. The application is responsible for ensuring enough space is reserved for the selector list to accommodate the specified buffer length.

**PhysBuf Block (PhysBuf)**

Address of the data structure. The PhysBuf Block is a variable length data structure. The first word is the Length of the PhysBuf Block in bytes. The remaining words of the structure are the selectors that address the physical video buffer. If Length is specified as 2, the required length of the PhysBuf Block is returned in its place.

**PhysBuf Block (USHORT)**

Length of PhysBuf structure in bytes

**selector (SEL)**

First selector

**selector (SEL)**

Next selector

**selector (SEL)**

... ..

**selector (SEL)**

Last selector

**Reserved (USHORT)** – input

Reserved word of 0s.

**rc (USHORT)** – return

Return code descriptions are:

<b>0</b>	<b>NO_ERROR</b>
<b>350</b>	<b>ERROR_VIO_PTR</b>
<b>429</b>	<b>ERROR_VIO_IN_BG</b>
<b>430</b>	<b>ERROR_VIO_ILLEGAL_DURING_POPUP</b>
<b>436</b>	<b>ERROR_VIO_INVALID_HANDLE</b>

465 ERROR\_VIO\_DETACHED  
494 ERROR\_VIO\_EXTENDED\_SG

## Remarks

If displaybuflen = 0, VioGetPhysBuf returns a selector that addresses the physical display buffer corresponding to the current mode. One selector is returned in Selector List. If a VioGetPhysBuf is issued after a VioGetBuf, then all VioWrtXX calls will no longer be written to the LVB. They will only be written to the physical display buffer. An application uses VioGetPhysBuf to get addressability to the physical display buffer. The selector returned by VioGetPhysBuf may be used only when an application program is executing in the foreground. When an application wants to access the physical display buffer, the application must call VioScrLock. VioScrLock either waits until the program is running in the foreground or returns a warning when the program is running in the background. For more information refer to "VioScrLock — Lock Screen" on page 5-50 and "VioScrUnLock — Unlock Screen" on page 5-60.

The buffer range specified for the physical screen buffer must fall between hex 'A0000' and 'BFFFF' inclusive. An application may issue VioGetPhysBuf only when it is running in the foreground. An application may issue VioGetPhysBuf more than once.

## C Language

```
typedef struct _VIOPHYSBUF { /* viopb */
    PBYTE pBuf; /* Buffer start address */
    ULONG cb; /* Buffer length */
    SEL asel[1]; /* Selector list */
} VIOPHYSBUF;

#define INCL_VIO

USHORT rc = VioGetPhysBuf(Structure, Reserved);

PVIOPHYSBUF Structure; /* Data structure */
USHORT Reserved; /* Reserved (must be zero) */

USHORT rc; /* return code */
```

## Assembler Language

```
VIOPHYSBUF struc
    viopb_pBuf dd ? ;Buffer start address
    viopb_cb dd ? ;buffer length
    viopb_asel dw 1 dup (?) ;selector list
VIOPHYSBUF ends

EXTRN VioGetPhysBuf:FAR
INCL_VIO EQU 1

PUSH@ OTHER Structure ;Data structure
PUSH WORD Reserved ;Reserved (must be zero)
CALL VioGetPhysBuf

Returns WORD
```

---

This call returns the current settings of the palette registers, overscan (border) color, blink/background intensity switch, color registers, underline location, or target VioSetMode display configuration.

<b>VioGetState (RequestBlock, VioHandle)</b>
--

## Parameters

**RequestBlock** (*PVOID*) — input/output

Address of the video state structures consisting of six different structures depending on the request type:

Type	Definition
0	Get palette registers
1	Get overscan (border) color
2	Get blink/background intensity switch
3	Get color registers
4	Reserved
5	Get the scan line for underlining
6	Get target VioSetMode display configuration.
7	Reserved

The six structures, depending on request type, are:

### VIOPALSTATE

Applies to EGA, VGA, or IBM Personal System/2 Display Adapter.

**length** (*USHORT*) — input

Length of structure, including length.

**38**            Maximum valid value.

**type** (*USHORT*) — input

Request type 0 for palette registers.

**palette** (*USHORT*) — input

First palette register in the palette register sequence; must be specified in the range 0 through 15. The palette registers are returned in sequential order. The number returned is based upon length.

**color** (*USHORT\*(length-6)/2*) — output

Color value for each palette register. The maximum number of entries in the color value array is 16.

### VIOOVERSCAN

Applies to CGA, VGA, or IBM Personal System/2 Display Adapter.

**length** (*USHORT*) — input

Length of structure, including length.

**6**            Only valid value.

**type** (*USHORT*) — input

Request type 1 for overscan (border) color.

**color** (*USHORT*) — input

Color value.

### VIOINTENSITY

Applies to CGA, EGA, MCGA, VGA, or IBM Personal System/2 Display Adapter.

**length** (*USHORT*) — input

Length of structure, including length.

**6**            Only valid value.

**type** (*USHORT*) — input  
Request type 2 for blink/background intensity switch.

**switch** (*USHORT*) — output  
Switch set as:

Value	Definition
0	Blinking foreground colors enabled.
1	High intensity background colors enabled.

## VIOCOLORREG

Applies to VGA, or IBM Personal System/2 Display Adapter.

**length** (*USHORT*) — input  
Length of structure, including length.

12 Length in bytes.

**type** (*USHORT*) — input  
Request type 3 for color registers.

**first color** (*USHORT*) — input  
First color register to get in the color register sequence; must be specified in the range 0 through 255. The color registers are returned in sequential order.

**number color** (*USHORT*) — input  
Number of color registers to get; must be specified in the range 1 through 256.

**dataarea** (*PCH*) — input  
Far address of a data area where the color registers are returned. The size of the data area must be three bytes times the number of color registers to get. The format of each entry returned is as follows:

<b>Byte 1</b>	Red value
<b>Byte 2</b>	Green value
<b>Byte 3</b>	Blue value

## VIOSETLINELOC

Applies to EGA, VGA, or IBM Personal System/2 Display Adapter.

**length** (*USHORT*) — input  
Length of structure, including length.

6 Length in bytes.

**type** (*USHORT*) — input  
Request type 5 to get the scan line for underlining. Underlining is enabled only when the foreground color is 1 or 9.

**scanline** (*USHORT*) — output  
The value returned is in the range 0 through 31 and is the scan line minus 1. A value of 32 means underlining is disabled.

## VIOSETTARGET

**length** (*USHORT*) — input  
Length of structure, including length.

6 Length in bytes.

**type** (*USHORT*) — input  
Request type 6 to get display configuration selected to be the target of the next VioSetMode.

**select** (*USHORT*) — output  
Configuration:

Value	Definition
0	Default selection algorithm. See VioSetMode.
1	Primary
2	Secondary.



# VioGetState — Get Video State

FAPI xWPM

**VioHandle** (*HVIO*) — input  
Reserved word of 0s.

**rc** (*USHORT*) — return  
Return code descriptions are:

0	NO_ERROR
355	ERROR_VIO_MODE
421	ERROR_VIO_INVALID_PARMS
436	ERROR_VIO_INVALID_HANDLE
438	ERROR_VIO_INVALID_LENGTH
465	ERROR_VIO_DETACHED
494	ERROR_VIO_EXTENDED_SG

## Family API Considerations

Request type = 6, Get Target VioSetMode Display Configuration, and request type = 5, Get Underline Location, are not supported in the family API.

**C Language**

```

typedef struct _VIOPALSTATE {
    USHORT cb;                /* Length of this structure */
    USHORT type;              /* Request type=0 get palette registers */
    USHORT iFirst;            /* First palette register to return */
    USHORT acolor[1];         /* Color value palette register */
}VIOPALSTATE;
typedef VIOPALSTATE far *PVIOPALSTATE;

typedef struct _VIOOVERSCAN {
    USHORT cb;                /* Length of this structure */
    USHORT type;              /* Request type=1 get overscan (border) color */
    USHORT color;             /* Color value */
}VIOOVERSCAN;
typedef VIOOVERSCAN far *PVIOOVERSCAN;

typedef struct _VIOINTENSITY {
    USHORT cb;                /* Length of this structure */
    USHORT type;              /* Request type=2 get blink/background
                             intensity switch */
    USHORT fs;                /* Value of blink/background switch */
}VIOINTENSITY;
typedef VIOINTENSITY far *PVIOINTENSITY;

typedef struct _VIOCOLORREG { /* viocreg */
    USHORT cb;
    USHORT type;
    USHORT firstcolorreg;
    USHORT numcolorregs;
    PCH colorregaddr;
}VIOCOLORREG;
typedef VIOCOLORREG far *PVIOCOLORREG;

typedef struct _VIOSETLINELOC { /* violine */
    USHORT cb;
    USHORT type;
    USHORT scanline;
}VIOSETLINELOC;
typedef VIOSETLINELOC far *PVIOSETLINELOC;

typedef struct _VIOSETTARGET { /* viosett */
    USHORT cb;
    USHORT type;
    USHORT defaultalgorithm;
}VIOSETTARGET;
typedef VIOSETTARGET far *PVIOSETTARGET;

#define INCL_VIO

USHORT rc = VioGetState(RequestBlock, VioHandle);

PVOID      RequestBlock; /* Request block */
HVIO       VioHandle;    /* Vio handle */

USHORT     rc;           /* return code */

```

# VioGetState — Get Video State

FAPI xWPM

## Assembler Language

```
VIOPALSTATE struc
    viopal_cb          dw ? ;Length of this structure in bytes
    viopal_type        dw ? ;Request type=0 get palette registers
    viopal_iFirst      dw ? ;First palette register to return
    viopal_acolor      dw 1 dup (?) ;Color value palette register
VIOPALSTATE ends

VIOOVERSCAN struc
    vioos_cb          dw ? ;Length of this structure
    vioos_type         dw ? ;Request type=1 get overscan (border) color
    vioos_color        dw ? ;Color value
VIOOVERSCAN ends

VIOINTENSITY struc
    vioint_cb         dw ? ;Length of this structure
    vioint_type        dw ? ;Request type=2 get blink/background intensity switch
    vioint_fs         dw ? ;Value of blink/background switch
VIOINTENSITY ends

VIOCOLORREG struc
    viocreg_cb        dw ? ;
    viocreg_type       dw ? ;
    viocreg_firstcolorreg dw ? ;
    viocreg_numcolorregs dw ? ;
    viocreg_colorregaddr dd ? ;
VIOCOLORREG ends

VIOSETULINELOC struc
    viouline_cb       dw ? ;
    viouline_type     dw ? ;
    viouline_scanline dw ? ;
VIOSETULINELOC ends

VIOSETTARGET struc
    viosett_cb        dw ? ;
    viosett_type      dw ? ;
    viosett_defaultalgorithm dw ? ;
VIOSETTARGET ends

EXTRN VioGetState:FAR
INCL_VIO EQU 1

PUSH@ OTHER RequestBlock ;Request block
PUSH WORD VioHandle ;Vio handle
CALL VioGetState

Returns WORD
```

# VioGlobalReg – Globally register a video subsystem

VioGlobalReg allows a subsystem to receive notification at the completion of VIO calls issued by all applications running in full-screen sessions.

**VioGlobalReg (ModuleName, EntryPoint, FunctionMask1, FunctionMask2, 0)**

## Parameters

**ModuleName (PSZ)** – input

Address of the ASCIIZ string containing the 1 – 8 character file name of the subsystem. The maximum length of the ASCIIZ string is 9 bytes including the terminating byte of zero. The module must be a dynamic link library but the name supplied must not include the .DLL extension.

**EntryPoint (PSZ)** – input

Address of the ASCIIZ name string containing the dynamic link entry point name of the routine in the subsystem to receive control when any of the registered functions is called. The maximum length of the ASCIIZ string is 33 bytes including the terminating byte of zero.

**FunctionMask1 (ULONG)** – input

A bit mask where each bit identifies a video function being registered. The bit definitions are shown below. The first word pushed onto the stack contains the high-order 16 bits of the function mask, and the second word contains the low-order 16 bits.

Bit	Registered Function	Bit	Registered Function
31	VioPrtScToggle	15	VioWrtCharStr
30	VioEndPopUp	14	VioWrtTTY
29	VioPopUp	13	VioWrtNCell
28	VioSavRedrawUndo	12	VioWrtNAttr
27	VioSavRedrawWait	11	VioWrtNChar
26	VioScrUnLock	10	VioReadCellStr
25	VioScrLock	9	VioReadCharStr
24	VioPrtSc	8	VioShowBuf
23	VioGetAnsi	7	VioSetMode
22	VioSetAnsi	6	VioSetCurType
21	VioScrollRt	5	VioSetCurPos
20	VioScrollLf	4	VioGetPhysBuf
19	VioScrollDn	3	VioGetBuf
18	VioScrollUp	2	VioGetMode
17	VioWrtCellStr	1	VioGetCurType
16	VioWrtCharStrAtt	0	VioGetCurPos

**FunctionMask2 (ULONG)** – input

A bit mask where each bit identifies a video function being registered. The bit mask has the format shown below. The first word pushed onto the stack contains the high order 16 bits of the function mask, and the second word contains the low order 16 bits. Unused bits are reserved and must be set to zero.

Bit	Registered Function
31 – 11	Reserved, must be set to zero.
10	VioDeRegister
9	VioRegister
8	VioSetState
7	VioGetState
6	VioSetFont
5	VioGetCp
4	VioSetCp
3	VioGetConfig
2	VioGetFont

# VioGlobalReg —

## Globally register a video subsystem

1 VioModeUndo  
0 VioModeWait

**Reserved (LONG)** — input  
Reserved and must be zero.

**rc (USHORT)** — return  
Return code descriptions are:

0 NO\_ERROR  
349 ERROR\_VIO\_INVALID\_MASK  
403 ERROR\_VIO\_INVALID\_ASCII  
426 ERROR\_VIO\_REGISTER  
494 ERROR\_VIO\_EXTENDED\_SG

### Remarks

Notification of VIO calls issued within the hard error handler and DOS (real mode) sessions is not provided.

When control is routed to EntryPoint, the stack appears as it did after the original VIO call except that four additional values have been pushed onto the stack. The first is the index number (WORD) of the routine called. The second is a near pointer (WORD). The third is the caller's DS register (WORD). The fourth is the return address (DWORD) to the VIO router.

For example, if VioSetCurPos were a registered function, the stack would appear as if the following instruction sequence were executed if VioSetCurPos were called and control routed to EntryPoint:

PUSH	WORD	Row
PUSH	WORD	Column
PUSH	WORD	VioHandle
CALL	FAR	VioSetCurPos
PUSH	WORD	Index
CALL	NEAR	Entry point in Vio router
PUSH	WORD	Caller's DS
CALL	FAR	Dynamic link entry point

The index numbers that correspond to the registered functions are listed below:

0 VioGetPhysBuf	22 VioSetAnsi
1 VioGetBuf	23 VioGetAnsi
2 VioShowBuf	24 VioPrtSc
3 VioGetCurPos	25 VioScrLock
4 VioGetCurType	26 VioScrUnLock
5 VioGetMode	27 VioSavRedrawWait
6 VioSetCurPos	28 VioSavRedrawUndo
7 VioSetCurType	29 VioPopUp
8 VioSetMode	30 VioEndPopUp
9 VioReadCharStr	31 VioPrtScToggle
10 VioReadCellStr	32 VioModeWait
11 VioWrtNChar	33 VioModeUndo
12 VioWrtNAttr	34 VioGetFont
13 VioWrtNCell	35 VioGetConfig
14 VioWrtCharStr	36 VioSetCp
15 VioWrtCharStrAtt	37 VioGetCp
16 VioWrtCellStr	38 VioSetFont
17 VioWrtTTY	39 VioGetState
18 VioScrollUp	40 VioSetState
19 VioScrollDn	41 VioRegister
20 VioScrollLf	42 VioDeRegister
21 VioScrollRt	

# VioGlobalReg – Globally register a video subsystem

On entry to the global subsystem, AX contains the return code that is returned to the application that issued the VIO call. The global subsystem must return with all stack parameters and all general purpose registers, including AX, restored to the same values as on entry.

All VIO functions within a session are serialized on a thread basis. That is, when a global subsystem receives control, it can safely assume that it is not called again from the same session until the current call has completed. Note, however, that VIO calls across different sessions are not serialized.

VioGlobalReg may only be issued during system initialization. After system initialization, VioGlobalReg returns ERROR\_VIO\_REGISTER. A globally registered subsystem is active for the life of the system.

If multiple global subsystems are registered, they are given control in the order that they are registered.

A globally registered subsystem receives control on VIO calls issued from all full-screen sessions except the hard error handler and DOS (real mode) sessions.

## C Language

```
#define INCL_VIO

USHORT rc = VioGlobalReg(ModuleName, EntryPoint, FunctionMask1, FunctionMask2, 0);

PSZ      ModuleName;      /* Module name */
PSZ      EntryPoint;      /* Entry point name */
ULONG    FunctionMask1;   /* Function mask 1 */
ULONG    FunctionMask2;   /* Function mask 2 */
LONG     0;               /* Reserved (must be zero) */

USHORT    rc;             /* return code */
```

## Assembler Language

```
EXTRN VioGlobalReg:FAR
INCL_VIO EQU 1

PUSH@ ASCIIZ ModuleName ;Module name
PUSH@ ASCIIZ EntryPoint ;Entry point name
PUSH DWORD FunctionMask1 ;Function mask 1
PUSH DWORD FunctionMask2 ;Function mask 2
PUSH DWORD 0 ;Reserved (must be zero)
CALL VioGlobalReg

Returns WORD
```

This call allows one thread within a process to cancel a VioModeWait issued by another thread within the same process.

**VioModeUndo (OwnerIndic, KillIndic, Reserved)**

## Parameters

**OwnerIndic (USHORT)** — input

Indicates whether the thread issuing VioModeUndo wants ownership of VioModeWait to be reserved for its process.

Value	Definition
0	Reserve ownership
1	Give up ownership.

**KillIndic (USHORT)** — input

Indicates whether the thread (with the outstanding VioModeWait) should be returned an error code or be terminated.

Value	Definition
0	Return error code
1	Terminate thread.

**Reserved (USHORT)** — input

Reserved word of 0s.

**rc (USHORT)** — return

Return code descriptions are:

0	NO_ERROR
421	ERROR_VIO_INVALID_PARMS
422	ERROR_VIO_FUNCTION_OWNED
427	ERROR_VIO_NO_MODE_THREAD
430	ERROR_VIO_ILLEGAL_DURING_POPUP
465	ERROR_VIO_DETACHED
486	ERROR_VIO_BAD_RESERVE
494	ERROR_VIO_EXTENDED_SG

## Remarks

VioModeUndo may be issued only by a thread within the process that owns VioModeWait. The thread issuing VioModeUndo can either reserve ownership of the VioModeWait function for its process or give up ownership. The thread whose VioModeWait is cancelled is optionally terminated.

## C Language

```
#define INCL_VIO
```

```
USHORT rc = VioModeUndo(OwnerIndic, KillIndic, Reserved);
```

```
USHORT OwnerIndic; /* Ownership indicator */
USHORT KillIndic; /* Terminate indicator */
USHORT Reserved; /* Reserved (must be zero) */

USHORT rc; /* return code */
```

**Assembler Language**

```
EXTRN VioModeUndo:FAR
INCL_VIO      EQU 1

PUSH WORD OwnerIndic ;Ownership indicator
PUSH WORD KillIndic  ;Terminate indicator
PUSH WORD Reserved   ;Reserved (must be zero)
CALL VioModeUndo
```

Returns WORD



This call allows a graphics mode application to be notified when it must restore its video mode, state, and modified display adapter registers. The return from this function call provides the notification.

**VioModeWait (RequestType, NotifyType, Reserved)**

## Parameters

**RequestType** (*USHORT*) — input

Application request event. RequestType = 0 indicates the application wants to be notified at the end of a pop-up to restore its mode. RequestType = 0 is the only event supported by VioModeWait.

**NotifyType** (*PUSHORT*) — output

Address of the operation to be performed by the application returning from VioModeWait. NotifyType = 0, indicating restore mode, is the only type of notification returned.

**Reserved** (*USHORT*) — input

Reserved word of 0s.

**rc** (*USHORT*) — return

Return code descriptions are:

0	NO_ERROR
421	ERROR_VIO_INVALID_PARMS
422	ERROR_VIO_FUNCTION_OWNED
423	ERROR_VIO_RETURN
424	ERROR_SCS_INVALID_FUNCTION
428	ERROR_VIO_NO_SAVE_RESTORE_THD
430	ERROR_VIO_ILLEGAL_DURING_POPUP
465	ERROR_VIO_DETACHED
494	ERROR_VIO_EXTENDED_SG

## Remarks

At the completion of an application or hard error pop-up (reference VioPopUp), OS/2 notifies the session that was originally interrupted for the pop-up to restore its mode. The return from this function call provides that notification. The thread that issued the call must perform the restore and then immediately re-issue VioModeWait.

When an application's VioModeWait thread is notified, the thread must restore its video mode, state, and modified display adapter registers. An application's VioModeWait thread does not restore the physical display buffer. OS/2 saves/restores the physical display buffer over a pop-up.

Only one process for a session can issue VioModeWait. The first process that issues VioModeWait becomes the owner of this function. (Refer to "VioModeUndo — Restore Mode Undo" on page 5-30.)

An application must issue VioModeWait only if it writes directly to the registers on the display adapter. Otherwise, the application can allow OS/2 to perform the required restore by not issuing VioModeWait.

When an application issues VioModeWait, it is also required to issue VioSavRedrawWait to be notified at screen switch time to perform a full save or restore. Reference "VioSavRedrawWait — Screen Save Redraw Wait" on page 5-48. Two application threads must be dedicated to performing these operations.

**C Language**

```
#define INCL_VIO

USHORT rc = VioModeWait(RequestType, NotifyType, Reserved);

USHORT      RequestType;  /* Request type */
PUSHORT      NotifyType;  /* Notify type (returned) */
USHORT      Reserved;     /* Reserved (must be zero) */

USHORT      rc;           /* return code */
```

**Assembler Language**

```
EXTRN VioModeWait:FAR
INCL_VIO      EQU 1

PUSH WORD    RequestType ;Request type
PUSH@ WORD   NotifyType  ;Notify type (returned)
PUSH WORD    Reserved    ;Reserved (must be zero)
CALL VioModeWait

Returns WORD
```

# VioPopUp – Allocate a Pop-up Display Screen

This call is issued by an application process when it requires a temporary screen to display a momentary message to the user.

**VioPopUp (Options, VioHandle)**

## Parameters

**Options (USHORT)** – input

Address of the bit flags that indicate which options to the application are being selected.

Bit	Description
15 – 2	Reserved, set to zero.
1	0 = Non-transparent operation. The video mode is set to text-mode 3, 3*, 3+, 7, or 7+. The highest resolution supported by the primary display adapter configured in the system is selected. The screen is cleared, and the cursor is positioned in the upper left corner of the display.  1 = Transparent operation. If the video mode of the outgoing foreground session is text (mode 2, 3, 7, or one of the * or + variations of these modes), no mode change occurs. The screen is not cleared, and the cursor remains in its current position. If transparent operation is selected, and if the video mode of the outgoing foreground session is not text (or if the outgoing foreground session has a VioSavRedrawWait thread), the pop-up request is refused. A unique error code ERROR_VIO_TRANSPARENT_POPUP is returned in this case.  OS/2 is responsible for saving and restoring the physical display buffer of the previous foreground session around a pop-up. This is true whether transparent or non-transparent operation is selected.
0	0 = Return with unique error code ERROR_VIO_EXISTING_POPUP if pop-up is not immediately available.  1 = Wait if pop-up is not immediately available.

**VioHandle (HVIDEO)** – input

Reserved words of 0s.

**rc (USHORT)** – return

Return code descriptions are:

0	NO_ERROR
405	ERROR_VIO_NO_POPUP
406	ERROR_VIO_EXISTING_POPUP
483	ERROR_VIO_TRANSPARENT_POPUP

## Remarks

VioPopUp is normally issued by the application when it is running in the background and wishes to immediately display a message to the user without waiting to become the active foreground session.

When an application process issues VioPopUp, it should wait for the return from the request. If the process allows any of its threads to write to the screen before VioPopUp returns a successful return code, the screen output may be directed to the application's normal video buffer rather than to the pop-up screen. If the process allows any of its threads to issue keyboard or mouse calls before VioPopUp returns a successful return code, the input is directed from the application's normal session. Once the process that issued VioPopUp receives a successful return code, video and keyboard calls issued by any of the threads in the pop-up process are directed to the pop-up screen. This continues until the process issues VioEndPopUp. At that time video and keyboard calls resume being directed to the application's normal video buffer.

## VioPopUp – Allocate a Pop-up Display Screen

There may be only one pop-up in existence at any time. If a process requests a pop-up and a pop-up already exists, the process has the choice of waiting for the prior pop-up to complete or receiving an immediate return with an error code. The error code indicates that the operation failed due to an existing pop-up having captured the screen.

Video pop-ups provide a mechanism for a background application to notify the operator of an abnormal event the operator must take some action. When considering the suitability of using pop-ups in a particular situation, the possible disruptive effect of pop-ups to the operator should be considered. If the operator were interrupted frequently by pop-ups issued by background applications, the operator would not effectively work with the foreground application.

While a video pop-up is in the foreground, the operator cannot use the hot key to switch to another application or to the shell. Before the operator can switch another application or the shell to the foreground, the pop-up application must issue VioEndPopUp.

While a video pop-up is in effect, all video calls from the previous foreground session are blocked until the process that issued VioPopUp issues VioEndPopUp.

When VioPopUp is issued, only the process within the session that issued VioPopUp is brought to the foreground. Assuming the session was already the foreground session, any video calls issued by other processes in that session are blocked until the process that issued VioPopUp issues VioEndPopUp.

DosExecPgm may not be issued by a process during a pop-up. The following video calls are the only calls that may be issued during the pop-up by the process that issued VioPopUp:

VioEndPopUp	VioScrollLf
VioGetConfig	VioSetCurPos
VioGetCp	VioSetCurType
VioGetFont	VioSetCp
VioGetAnsi	VioSetFont
VioGetState	VioSetState
VioGetCurPos	VioWrtNChar
VioGetCurType	VioWrtNAttr
VioGetMode	VioWrtNCell
VioReadCharStr	VioWrtCharStr
VioReadCellStr	VioWrtCharStrAtt
VioScrollRt	VioWrtCellStr
VioScrollUp	VioWrtTTY
VioScrollDn	

Selectors to the physical display buffer that the issuing process obtained on a prior VioGetPhysBuf call may not be used during the pop-up.

When an application registers a replacement for VioPopUp within a session, the registered routine is invoked only when that session is in the foreground. If VioPopUp is issued when that session is in the background, the OS/2 default routine is invoked. If the application's session is using a keyboard or mouse monitor, the monitor does not intercept data while the pop-up is active.

### PM Considerations

This function can be used from within a PM application. Kbdxxx, Mouxxx, and Vioxxx calls (with a zero handle) are all allowed between VioPopUp and VioEndPopUp, and are directed to the pop-up screen. An error is returned if issued with a non-zero handle.

# VioPopUp – Allocate a Pop-up Display Screen

## C Language

```
#define INCL_VIO

USHORT rc = VioPopUp(Options, VioHandle);

PUSHORT Options; /* Option Flags */
HVIO VioHandle; /* Vio handle */

USHORT rc; /* return code */
```

## Assembler Language

```
EXTRN VioPopUp:FAR
INCL_VIO EQU 1

PUSH@ WORD Options ;Option Flags
PUSH WORD VioHandle ;Vio handle
CALL VioPopUp

Returns WORD
```

---

This call is issued by the Session Manager when the operator presses PrtSc.

<b>VioPrtSc (VioHandle)</b>
-----------------------------

## Parameters

**VioHandle (HVIO)** – input  
Reserved word of 0s.

**rc (USHORT)** – return  
Return code descriptions are:

<b>0</b>	<b>NO_ERROR</b>
<b>355</b>	<b>ERROR_VIO_MODE</b>
<b>402</b>	<b>ERROR_VIO_SMG_ONLY</b>
<b>436</b>	<b>ERROR_VIO_INVALID_HANDLE</b>
<b>465</b>	<b>ERROR_VIO_DETACHED</b>

## Remarks

VioPrtSc supports text modes 0 through 3, and 7. An Alternate Video Subsystem may want to register a replacement for VioPrtSc. An advanced video subsystem could set a graphics mode while the mode known to the base video subsystem PrtSc routine is text. Then, if the operator presses PrtSc, the printer output is unpredictable. VioPrtSc is reserved for use by the session manager. Application programs may not issue VioPrtSc.

Three beeps are generated if a hard error is detected while writing to the printer.

## C Language

```
#define INCL_VIO

USHORT rc = VioPrtSc(VioHandle);

HVIO      VioHandle;    /* Vio handle */

USHORT     rc;          /* return code */
```

## Assembler Language

```
EXTRN VioPrtSc:FAR
INCL_VIO      EQU 1

PUSH WORD VioHandle ;Vio handle
CALL VioPrtSc

Returns WORD
```

---

This call is called by the Session Manager when the operator presses Ctrl and PrtSc.

<b>VioPrtScToggle (VioHandle)</b>
-----------------------------------

## Parameters

**VioHandle (HVIO)** — input  
Reserved word of 0s.

**rc (USHORT)** — return  
Return code descriptions are:

0	NO_ERROR
355	ERROR_VIO_MODE
402	ERROR_VIO_SMG_ONLY
430	ERROR_VIO_ILLEGAL_DURING_POPUP
436	ERROR_VIO_INVALID_HANDLE
465	ERROR_VIO_DETACHED

## Remarks

VioPrtScToggle toggles the Ctrl and PrtSc state of the foreground session. When the Ctrl and PrtSc state is on, all VioWrtTTY calls from that session are echoed to the print device.

VioPrtScToggle can only be called by the session manager. If an application issues this call, it receives an error code.

Three beeps are generated if a hard error is detected while writing to the printer. When Ctrl and PrtSc is turned off, the operator may have to press the Enter key to cause output spooled while Ctrl and PrtSc was active to be printed. This is because the spool files are closed when the next VioWrtTTY is executed in the session, such as writing the prompt in response to the Enter key.

## C Language

```
#define INCL_VIO

USHORT rc = VioPrtScToggle(VioHandle);

HVIO      VioHandle;    /* Vio handle */

USHORT    rc;           /* return code */
```

## Assembler Language

```
EXTRN VioPrtScToggle:FAR
INCL_VIO      EQU 1

PUSH WORD    VioHandle    ;Vio handle
CALL VioPrtScToggle

Returns WORD
```

This call reads a string of character-attribute pairs (cells) from the screen, starting at the specified location.

<b>VioReadCellStr</b> ( <b>CellStr</b> , <b>Length</b> , <b>Row</b> , <b>Column</b> , <b>VioHandle</b> )
--

## Parameters

**CellStr** (*PCH*) – output

Address of the buffer where the cell string is returned.

**Length** (*PUSHORT*) – input/output

Address of the buffer length in bytes. Length must take into account that each character-attribute(s) entry in the buffer is 2 or 4 bytes. If the length of the buffer is not sufficient, the last entry is not complete.

**Row** (*USHORT*) – input

Starting row of the field to read, 0 is the top row.

**Column** (*USHORT*) – input

Starting column of the field to read, 0 is the leftmost column.

**VioHandle** (*HVIO*) – input

This must be zero unless the caller is a Presentation Manager application, in which case it must be the value returned by VioGetPs.

**rc** (*USHORT*) – return

Return code descriptions are:

<b>0</b>	<b>NO_ERROR</b>
<b>355</b>	<b>ERROR_VIO_MODE</b>
<b>358</b>	<b>ERROR_VIO_ROW</b>
<b>359</b>	<b>ERROR_VIO_COL</b>
<b>436</b>	<b>ERROR_VIO_INVALID_HANDLE</b>
<b>465</b>	<b>ERROR_VIO_DETACHED</b>

## Remarks

If a string read comes to the end of the line and is not complete, the string read continues at the beginning of the next line. If the read comes to the end of the screen and is not complete, the read terminates and the length is set to the length of the buffer that was filled.

## PM Considerations

VioReadCellStr reads a string of character/attributes (or cells) from the Advanced VIO presentation space starting at the specified location.

## C Language

```
#define INCL_VIO
```

```
USHORT rc = VioReadCellStr(CellStr, Length, Row, Column, VioHandle);
```

```
PCH      CellStr;      /* Cell string buffer */
PUSHORT  Length;      /* Length of cell string buffer */
USHORT   Row;         /* Starting row location */
USHORT   Column;      /* Starting column location */
HVIO     VioHandle;    /* Video handle */

USHORT   rc;          /* return code */
```



# VioReadCellStr — Read Char/Attr String

FAPI

## Assembler Language

```
EXTRN VioReadCellStr:FAR  
INCL_VIO EQU 1
```

```
PUSH@ OTHER CellStr ;Cell string buffer  
PUSH@ WORD Length ;Length of cell string buffer  
PUSH WORD Row ;Starting row location  
PUSH WORD Column ;Starting column location  
PUSH WORD VioHandle ;Video handle  
CALL VioReadCellStr
```

Returns WORD

This call reads a string of characters from the display starting at the specified location.

**VioReadCharStr** (*CharStr*, *Length*, *Row*, *Column*, *VioHandle*)

## Parameters

**CharStr** (*PCH*) — output

Address of the buffer where the character string is returned.

**Length** (*PUSHORT*) — input/output

Address of the buffer length in bytes.

**Row** (*USHORT*) — input

Starting row of the field to read, 0 is the top row.

**Column** (*USHORT*) — input

Starting column of the field to read, 0 is the leftmost column.

**VioHandle** (*HVIO*) — input

This must be zero unless the caller is a Presentation Manager application, in which case it must be the value returned by VioGetPs.

**rc** (*USHORT*) — return

Return code descriptions are:

<b>0</b>	<b>NO_ERROR</b>
<b>355</b>	<b>ERROR_VIO_MODE</b>
<b>358</b>	<b>ERROR_VIO_ROW</b>
<b>359</b>	<b>ERROR_VIO_COL</b>
<b>436</b>	<b>ERROR_VIO_INVALID_HANDLE</b>
<b>465</b>	<b>ERROR_VIO_DETACHED</b>

## Remarks

If a string read comes to the end of the line and is not complete, then the string read continues at the beginning of the next line. If the read comes to the end of the screen and is not complete, the read terminates and the length is set to the number of characters read.

## PM Considerations

VioReadCharStr reads a character string from the Advanced VIO presentation space starting at the specified location.

## C Language

```
#define INCL_VIO
```

```
USHORT rc = VioReadCharStr(CharStr, Length, Row, Column, VioHandle);
```

```
PCH          CharStr;      /* Character buffer */
PUSHORT      Length;       /* Length of buffer */
USHORT       Row;          /* Starting row location */
USHORT       Column;       /* Starting column location */
HVIO         VioHandle;    /* Video handle */

USHORT       rc;           /* return code */
```

# VioReadCharStr — Read Character String

FAPI

## Assembler Language

```
EXTRN VioReadCharStr:FAR
INCL_VIO EQU 1

PUSH@ OTHER CharStr ;Character buffer
PUSH@ WORD Length ;Length of buffer
PUSH WORD Row ;Starting row location
PUSH WORD Column ;Starting column location
PUSH WORD VioHandle ;Video handle
CALL VioReadCharStr
```

Returns WORD

This call registers an alternate video subsystem within a session.

**VioRegister (ModuleName, EntryPoint, FunctionMask1, FunctionMask2)**

## Parameters

**ModuleName (PSZ)** – input

Address of the ASCIIZ string containing the 1 – 8 character file name of the subsystem. The maximum length of the ASCIIZ string is 9 bytes including the terminating byte of zero. The module must be a dynamic link library but the name supplied must not include the .DLL extension.

**EntryPoint (PSZ)** – input

Address of the ASCIIZ name string containing the dynamic link entry point name of the routine in the subsystem to receive control when any of the registered functions is called. The maximum length of the ASCIIZ string is 33 bytes including the terminating byte of zero.

**FunctionMask1 (ULONG)** – input

A bit mask where each bit identifies a video function being registered. The bit definitions are shown below. The first word pushed onto the stack contains the high-order 16 bits of the function mask, and the second word contains the low-order 16 bits.

Bit	Registered Function	Bit	Registered Function
31	VioPrtScToggle	15	VioWrtCharStr
30	VioEndPopUp	14	VioWrtTTY
29	VioPopUp	13	VioWrtNCell
28	VioSavRedrawUndo	12	VioWrtNAttr
27	VioSavRedrawWait	11	VioWrtNChar
26	VioScrUnLock	10	VioReadCellStr
25	VioScrLock	9	VioReadCharStr
24	VioPrtSc	8	VioShowBuf
23	VioGetAnsi	7	VioSetMode
22	VioSetAnsi	6	VioSetCurType
21	VioScrollRt	5	VioSetCurPos
20	VioScrollLf	4	VioGetPhysBuf
19	VioScrollDn	3	VioGetBuf
18	VioScrollUp	2	VioGetMode
17	VioWrtCellStr	1	VioGetCurType
16	VioWrtCharStrAtt	0	VioGetCurPos

**FunctionMask2 (ULONG)** – input

A bit mask where each bit identifies a video function being registered. The bit mask has the format shown below. The first word pushed onto the stack contains the high order 16 bits of the function mask, and the second word contains the low order 16 bits. Unused bits are reserved and must be set to zero.

Bit	Description
31 – 9	Reserved, set to zero
8	VioSetState
7	VioGetState
6	VioSetFont
5	VioGetCp
4	VioSetCp
3	VioGetConfig
2	VioGetFont
1	VioModeUndo
0	VioModeWait

# VioRegister —

## Register Video Subsystem

xWPM

**rc** (*USHORT*) — return

Return code descriptions are:

0	NO_ERROR
349	ERROR_VIO_INVALID_MASK
403	ERROR_VIO_INVALID_ASCII
426	ERROR_VIO_REGISTER
430	ERROR_VIO_ILLEGAL_DURING_POPUP
465	ERROR_VIO_DETACHED
494	ERROR_VIO_EXTENDED_SG

### Remarks

An alternate video subsystem must register which video calls it handles. The default OS/2 video subsystem is the Base Video Subsystem.

When any of the registered functions are called, control is routed to EntryPoint. When this routine is entered, four additional values (5 words) are pushed onto the stack.

The first value is the index number (Word) of the routine being called. The second value is a near pointer (Word). The third value is the caller's DS register (Word). The fourth value is the return address (DWord) to the VIO router.

For example, if VioSetCurPos were a registered function, the stack would appear as if the following instruction sequence were executed if VioSetCurPos were called and control routed to EntryPoint:

PUSH	WORD	Row
PUSH	WORD	Column
PUSH	WORD	VioHandle
CALL	FAR	VioSetCurPos
PUSH	WORD	Index
CALL	NEAR	Entry point in Vio router
PUSH	WORD	Caller's DS
CALL	FAR	Dynamic link entry point

The index numbers that correspond to the registered functions are listed below:

0 VioGetPhysBuf	22 VioSetAnsi
1 VioGetBuf	23 VioGetAnsi
2 VioShowBuf	24 VioPrtSc
3 VioGetCurPos	25 VioScrLock
4 VioGetCurType	26 VioScrUnLock
5 VioGetMode	27 VioSavRedrawWait
6 VioSetCurPos	28 VioSavRedrawUndo
7 VioSetCurType	29 VioPopUp
8 VioSetMode	30 VioEndPopUp
9 VioReadCharStr	31 VioPrtScToggle
10 VioReadCellStr	32 VioModeWait
11 VioWrtNChar	33 VioModeUndo
12 VioWrtNAttr	34 VioGetFont
13 VioWrtNCell	35 VioGetConfig
14 VioWrtCharStr	36 VioSetCp
15 VioWrtCharStrAtt	37 VioGetCp
16 VioWrtCellStr	38 VioSetFont
17 VioWrtTTY	39 VioGetState
18 VioScrollUp	40 VioSetState
19 VioScrollDn	
20 VioScrollLf	
21 VioScrollRt	

## VioRegister — Register Video Subsystem

When a registered function returns to the video router, the return code is interpreted as follows:

- Return code = 0**                      No error. Do not invoke the corresponding Base Video Subsystem routine. Return to caller with Return code = 0.
- Return code = -1**                    No error. Invoke the corresponding Base Video Subsystem routine. Return to caller with Return code = return code from Base Video Subsystem.
- Return code = error (not 0 or -1)** Do not invoke the corresponding Base Video Subsystem routine. Return to caller with Return code = error.

When an application registers a replacement for VioPopUp within a session, the registered routine is only invoked when that session is in the foreground. If VioPopUp is issued when that session is in the background, the OS/2 default routine is invoked.

An alternate video subsystem should be designed so the routines registered do not cause any hard errors when they are invoked. Otherwise, a system lockout occurs. Code and data segments of registered routines, that might be loaded from diskette, must be preloaded.

All VIO functions within a session are serialized on a thread basis. That is, when an alternate video subsystem receives control, it can safely assume that it is not called again from the same session until the current call has completed.

### C Language

```
#define INCL_VIO

USHORT rc = VioRegister(ModuleName, EntryPoint, FunctionMask1,
                        FunctionMask2);

PSZ      ModuleName; /* Module name */
PSZ      EntryPoint; /* Entry point name */
ULONG    FunctionMask1; /* Function mask 1 */
ULONG    FunctionMask2; /* Function mask 2 */

USHORT    rc; /* return code */
```

### Assembler Language

```
EXTRN VioRegister:FAR
INCL_VIO EQU 1

PUSH@ ASCIIZ ModuleName ;Module name
PUSH@ ASCIIZ EntryPoint ;Entry point name
PUSH DWORD FunctionMask1 ;Function mask 1
PUSH DWORD FunctionMask2 ;Function mask 2
CALL VioRegister

Returns WORD
```

# VioSavRedrawUndo – Screen Save Redraw Undo

xWPM

This call allows one thread within a process to cancel a VioSavRedrawWait issued by another thread within the same process.

**VioSavRedrawUndo (OwnerIndic, KillIndic, VioHandle)**

## Parameters

**OwnerIndic (USHORT)** – input

Indicates whether the thread issuing VioSavRedrawUndo wants ownership of VioSavRedrawWait to be reserved for its process.

Value	Definition
0	Reserve ownership
1	Give up ownership.

**KillIndic (USHORT)** – input

Indicates whether the thread with the outstanding VioSavRedrawWait should be returned an error code or be terminated.

Value	Definition
0	Return error code
1	Terminate thread.

**VioHandle (HVIO)** – Input

Reserved word of 0s.

**rc (USHORT)** – return

Return code descriptions are:

0	NO_ERROR
421	ERROR_VIO_INVALID_PARMS
422	ERROR_VIO_FUNCTION_OWNED
428	ERROR_VIO_NO_SAVE_RESTORE_THD
430	ERROR_VIO_ILLEGAL_DURING_POPUP
465	ERROR_VIO_DETACHED
494	ERROR_VIO_EXTENDED_SG

## Remarks

The issuing thread can reserve ownership of VioSavRedrawWait for its process or give it up. The thread whose VioSavRedrawWait was cancelled is optionally terminated. VioSavRedrawUndo may be issued only by a thread within the same process that owns VioSavRedrawWait.

## C Language

```
#define INCL_VIO
```

```
USHORT rc = VioSavRedrawUndo(OwnerIndic, KillIndic, VioHandle);
```

```
USHORT OwnerIndic; /* Ownership indicator */
USHORT KillIndic; /* Terminate indicator */
HVIO VioHandle; /* Video handle */

USHORT rc; /* return code */
```

**Assembler Language**

```
EXTRN VioSavRedrawUndo:FAR  
INCL_VIO EQU 1
```

```
PUSH WORD OwnerIndic ;Ownership indicator  
PUSH WORD KillIndic ;Terminate indicator  
PUSH WORD VioHandle ;Video handle  
CALL VioSavRedrawUndo
```

Returns WORD



# VioSavRedrawWait — Screen Save Redraw Wait

xWPM

This call notifies a graphics mode application when it must save or redraw its screen image.

**VioSavRedrawWait (SavRedrawIndic, NotifyType, VioHandle)**

## Parameters

**SavRedrawIndic** (*USHORT*) — input

Indicates which events the application is waiting for:

Value	Definition
0	The session manager notifies the application for both save and redraw operations.
1	The session manager notifies the application for redraw operations only.

**NotifyType** (*PUSHORT*) — output

Address that specifies the operation to be performed by the application upon return from VioSavRedrawWait:

Value	Definition
0	Save screen image
1	Restore screen image.

**VioHandle** (*HVIO*) — input

Reserved word of 0s.

**rc** (*USHORT*) — return

Return code descriptions are:

0	NO_ERROR
421	ERROR_VIO_INVALID_PARMS
422	ERROR_VIO_FUNCTION_OWNED
423	ERROR_VIO_RETURN
430	ERROR_VIO_ILLEGAL_DURING_POPUP
436	ERROR_VIO_INVALID_HANDLE
465	ERROR_VIO_DETACHED
494	ERROR_VIO_EXTENDED_SG

## Remarks

OS/2 uses VioSavRedrawWait to notify a graphics mode application to save or restore its screen image at screen switch time. The application in the outgoing foreground session is notified to perform a save. The application in the incoming foreground session is notified to perform a restore. The application must perform the action requested and immediately re-issue VioSavRedrawWait. When an application performs a save, it saves its physical display buffer, video mode, and any other information the application needs to completely redraw its screen at restore time.

Only one process per session can issue VioSavRedrawWait. The process that first issues VioSavRedrawWait becomes the owner of the function.

A text mode application must issue VioSavRedrawWait only if the application writes directly to the registers on the display adapter. Assuming VioSavRedrawWait is not issued by a text mode application, OS/2 performs the required saves and restores.

An application that issues VioSavRedrawWait may also need to issue VioModeWait. This would allow the application to be notified when it must restore its mode at the completion of an application or hard error pop-up. Refer to "VioModeWait — Restore Mode Wait" on page 5-32 for more information. Two application threads would be required to perform these operations in this case.

At the time a VioSavRedrawWait thread is notified, the session is in transition to/from the background. Although the session's official status is background, any selector to the physical display buffer previously obtained by the VioSavRedrawWait process (through VioGetPhysBuf) is valid at this time. The physical

display buffer must be accessed without issuing VioScrLock. Since the session's official status is background, any thread waits if it issues VioScrLock with the "wait if unsuccessful" option.

An application containing a VioSavRedrawWait thread should be designed so that the process does not cause any hard errors while the VioSavRedrawWait thread is running, otherwise a system lockout may occur.

An application's VioSavRedrawWait thread may be notified to perform a restore before it is notified to perform a save. This happens if the application was running in the background the first time it issued VioSavRedrawWait. The return from this function call provides the notification. The thread that issues the call performs the save or redraw and then reissues VioSavRedrawWait to wait until its screen image must be saved or redrawn again.

## C Language

```
#define INCL_VIO
```

```
USHORT rc = VioSavRedrawWait(SavRedrawIndic, NotifyType, VioHandle);
```

```
USHORT      SavRedrawIndic; /* Save/redraw indicator */
PUSHORT     NotifyType;    /* Notify type (returned) */
HVIO        VioHandle;     /* Video handle */
```

```
USHORT      rc;            /* return code */
```

## Assembler Language

```
EXTRN VioSavRedrawWait:FAR
INCL_VIO EQU 1
```

```
PUSH WORD SavRedrawIndic ;Save/redraw indicator
PUSH0 WORD NotifyType ;Notify type (returned)
PUSH WORD VioHandle ;Video handle
CALL VioSavRedrawWait
```

```
Returns WORD
```

---

This call requests ownership of (locks) the physical display buffer.

<b>VioScrLock</b> ( <b>WaitFlag</b> , <b>Status</b> , <b>VioHandle</b> )
--

## Parameters

**WaitFlag** (*USHORT*) — input

Indicates whether the process should block until the screen I/O can take place.

Value	Definition
0	Return if screen I/O not available
1	Wait until screen I/O is available.

**Status** (*PUCHAR*) — output

Address of the Indicator of whether the lock is successful, described below.

Value	Definition
0	Lock successful
1	Lock unsuccessful (in the case of no wait).

Status is returned only when AX = 0.

Status = 1 may be returned only when WaitFlag = 0.

**VioHandle** (*HVIO*) — input

Reserved word of 0s.

**rc** (*USHORT*) — return

Return code descriptions are:

0	NO_ERROR
366	ERROR_VIO_WAIT_FLAG
430	ERROR_VIO_ILLEGAL_DURING_POPUP
434	ERROR_VIO_LOCK
436	ERROR_VIO_INVALID_HANDLE
465	ERROR_VIO_DETACHED
494	ERROR_VIO_EXTENDED_SG

## Remarks

This function call permits a process to determine if I/O to the physical screen buffer can take place. This prevents the process from writing to the physical buffer when the process is in the background. Processes must cooperate with the system in coordinating screen accesses.

Screen switching is disabled while the screen lock is in place. If a screen switch is suspended by a screen lock, and if the application holding the lock does not issue VioScrUnLock within a system-defined time limit, the screen switch occurs, and the process holding the lock is frozen in the background. A process should yield the screen lock as soon as possible to avoid being frozen when running in the background. The timeout on the lock does not begin until a screen switch is requested.

When the screen lock is in effect and another thread in the same or different process (in the same session) issues VioScrLock, the second thread receives an error code. VioScrUnLock must be issued by a thread within the same process that issued VioScrLock.

## Family API Considerations

Some options operate differently in the DOS mode than in the OS/2 mode. Therefore, the following restriction applies to VioScrLock when coding in the DOS mode:

The status always indicates the lock is successful (Return code = 0).

**C Language**

```
#define INCL_VIO

USHORT rc = VioScrLock(WaitFlag, Status, VioHandle);

USHORT      WaitFlag;      /* Block or not */
PUCHAR      Status;        /* Lock status (returned) */
HVIO        VioHandle;     /* Video handle */

USHORT      rc;            /* return code */
```

**Assembler Language**

```
EXTRN VioScrLock:FAR
INCL_VIO EQU 1

PUSH WORD WaitFlag ;Block or not
PUSH@ BYTE Status ;Lock status (returned)
PUSH WORD VioHandle ;Video handle
CALL VioScrLock

Returns WORD
```

# VioScrollDn – Scroll Screen Down

FAP1

---

This call scrolls the entire display buffer (or area specified within the display buffer) down.

<b>VioScrollDn</b> ( <b>TopRow</b> , <b>LeftCol</b> , <b>BotRow</b> , <b>RightCol</b> , <b>Lines</b> , <b>Cell</b> , <b>VioHandle</b> )
---

## Parameters

**TopRow** (*USHORT*) – input  
Top row to be scrolled.

**LeftCol** (*USHORT*) – input  
Left column to be scrolled.

**BotRow** (*USHORT*) – input  
Bottom row to be scrolled.

**RightCol** (*USHORT*) – input  
Right column to be scrolled.

**Lines** (*USHORT*) – input  
Number of lines to be inserted at the top of the screen area being scrolled. If 0 is specified, no lines are scrolled.

**Cell** (*PBYTE*) – input  
Address of the character-attribute(s) pair (2 or 4 bytes) used as a fill character on inserted lines.

**VioHandle** (*HVIO*) – input  
This must be zero unless the caller is a Presentation Manager application, in which case it must be the value returned by VioGetPs.

**rc** (*USHORT*) – return  
Return code descriptions are:

0	NO_ERROR
355	ERROR_VIO_MODE
358	ERROR_VIO_ROW
359	ERROR_VIO_COL
436	ERROR_VIO_INVALID_HANDLE
465	ERROR_VIO_DETACHED

## Remarks

TopRow = 0 and LeftCol = 0 identifies the top left corner of the screen.

If a value greater than the maximum value is specified for TopRow, LeftCol, BotRow, RightCol, or Lines, the maximum value for that parameter is used.

If TopRow and LeftCol = 0 and if BotRow, RightCol, and Lines = 65535 (or –1 in assembler language), the entire screen is filled with the character-attribute pair defined by Cell.

## C Language

```
#define INCL_VIO
```

```
USHORT rc = VioScrollDn(TopRow, LeftCol, BotRow, RightCol, Lines, Cell,  
                        VioHandle);
```

```
USHORT      TopRow;      /* Top row */  
USHORT      LeftCol;     /* Left column */  
USHORT      BotRow;     /* Bottom row */  
USHORT      RightCol;    /* Right column */  
USHORT      Lines;      /* Number of lines */  
PBYTE       Cell;       /* Cell to be written */  
HVIDEO      VioHandle;   /* Video handle */  
  
USHORT      rc;          /* return code */
```

## Assembler Language

```
EXTRN VioScrollDn:FAR  
INCL_VIO EQU 1
```

```
PUSH WORD TopRow      ;Top row  
PUSH WORD LeftCol     ;Left column  
PUSH WORD BotRow      ;Bottom row  
PUSH WORD RightCol    ;Right column  
PUSH WORD Lines       ;Number of lines  
PUSH@ OTHER Cell      ;Cell to be written  
PUSH WORD VioHandle   ;Video handle  
CALL VioScrollDn
```

```
Returns WORD
```

# VioScrollLf – Scroll Screen Left

FAP1

---

This call scrolls the entire display buffer (or area specified within the display buffer) to the left.

<b>VioScrollLf</b> ( <b>TopRow</b> , <b>LeftCol</b> , <b>BotRow</b> , <b>RightCol</b> , <b>Lines</b> , <b>Cell</b> , <b>VioHandle</b> )
---

## Parameters

**TopRow** (*USHORT*) – input

Top row to be scrolled.

**LeftCol** (*USHORT*) – input

Left column to be scrolled.

**BotRow** (*USHORT*) – input

Bottom row to be scrolled.

**RightCol** (*USHORT*) – input

Right column to be scrolled.

**Lines** (*USHORT*) – input

Number of columns to be inserted at the right of the screen area being scrolled. If 0 is specified, no lines are scrolled.

**Cell** (*PBYTE*) – input

Address of the character attribute(s) pair (2 or 4 bytes) used as a fill character on inserted columns.

**VioHandle** (*HVIO*) – input

This must be zero unless the caller is a Presentation Manager application, in which case it must be the value returned by VioGetPs.

**rc** (*USHORT*) – return

Return code descriptions are:

0	NO_ERROR
355	ERROR_VIO_MODE
358	ERROR_VIO_ROW
359	ERROR_VIO_COL
436	ERROR_VIO_INVALID_HANDLE
465	ERROR_VIO_DETACHED

## Remarks

TopRow = 0 and LeftCol = 0 identifies the top left corner of the screen.

If a value greater than the maximum value is specified for TopRow, LeftCol, BotRow, RightCol, or Lines, the maximum value for that parameter is used.

If TopRow and LeftCol = 0 and if BotRow, RightCol, and Lines = 65535 (or –1 in assembler language), the entire screen is filled with the character-attribute pair defined by Cell.

## C Language

```
#define INCL_VIO
```

```
USHORT rc = VioScrollLf(TopRow, LeftCol, BotRow, RightCol, Lines, Cell,  
                        VioHandle);
```

```
USHORT      TopRow;      /* Top row */  
USHORT      LeftCol;     /* Left column */  
USHORT      BotRow;     /* Bottom row */  
USHORT      RightCol;    /* Right column */  
USHORT      Lines;      /* Number of lines */  
PBYTE      Cell;        /* Cell to be written */  
HVID        VioHandle;   /* Video Handle */  
  
USHORT      rc;          /* return code */
```

## Assembler Language

```
EXTRN VioScrollLf:FAR  
INCL_VIO EQU 1
```

```
PUSH WORD TopRow      ;Top row  
PUSH WORD LeftCol     ;Left column  
PUSH WORD BotRow      ;Bottom row  
PUSH WORD RightCol    ;Right column  
PUSH WORD Lines       ;Number of lines  
PUSH@ OTHER Cell      ;Cell to be written  
PUSH WORD VioHandle   ;Video Handle  
CALL VioScrollLf
```

```
Returns WORD
```



# VioScrollRt – Scroll Screen Right

FAP1

This call scrolls the entire display buffer (or area specified within the display buffer) to the right.

<b>VioScrollRt</b> ( <i>TopRow</i> , <i>LeftCol</i> , <i>BotRow</i> , <i>RightCol</i> , <i>Lines</i> , <i>Cell</i> , <i>VioHandle</i> )
---

## Parameters

**TopRow** (*USHORT*) – input

Top row to be scrolled.

**LeftCol** (*USHORT*) – input

Left column to be scrolled.

**BotRow** (*USHORT*) – input

Bottom row to be scrolled.

**RightCol** (*USHORT*) – input

Right column to be scrolled.

**Lines** (*USHORT*) – input

Number of columns to be inserted at the left of the screen area being scrolled. If 0 is specified, no lines are scrolled.

**Cell** (*PBYTE*) – input

Address of the character attribute(s) pair (2 or 4 bytes) used as a fill character on inserted columns.

**VioHandle** (*HVIO*) – input

This must be zero unless the caller is a Presentation Manager application, in which case it must be the value returned by VioGetPs.

**rc** (*USHORT*) – return

Return code descriptions are:

0	NO_ERROR
355	ERROR_VIO_MODE
358	ERROR_VIO_ROW
359	ERROR_VIO_COL
436	ERROR_VIO_INVALID_HANDLE
465	ERROR_VIO_DETACHED

## Remarks

TopRow = 0 and LeftCol = 0 identifies the top left corner of the screen.

If a value greater than the maximum value is specified for TopRow, LeftCol, BotRow, RightCol, or Lines, the maximum value for that parameter is used.

If TopRow and LeftCol = 0 and if BotRow, RightCol, and Lines = 65535 (or –1 in assembler language), the entire screen is filled with the character-attribute pair defined by Cell.

## C Language

```
#define INCL_VIO

USHORT rc = VioScrollRt(TopRow, LeftCol, BotRow, RightCol, Lines, Cell,
                        VioHandle);

USHORT      TopRow;      /* Top row */
USHORT      LeftCol;     /* Left column */
USHORT      BotRow;      /* Bottom row */
USHORT      RightCol;    /* Right column */
USHORT      Lines;       /* Number of lines */
PBYTE      Cell;         /* Cell to be written */
HVIO      VioHandle;     /* Video handle */

USHORT      rc;          /* return code */
```

## Assembler Language

```
EXTRN VioScrollRt:FAR
INCL_VIO      EQU 1

PUSH WORD TopRow      ;Top row
PUSH WORD LeftCol     ;Left column
PUSH WORD BotRow      ;Bottom row
PUSH WORD RightCol    ;Right column
PUSH WORD Lines       ;Number of lines
PUSH@ OTHER Cell      ;Cell to be written
PUSH WORD VioHandle   ;Video handle
CALL VioScrollRt

Returns WORD
```

---

This call scrolls the entire display buffer (or area specified within the display buffer) up.

<b>VioScrollUp</b> ( <b>TopRow</b> , <b>LeftCol</b> , <b>BotRow</b> , <b>RightCol</b> , <b>Lines</b> , <b>Cell</b> , <b>VioHandle</b> )
---

## Parameters

**TopRow** (*USHORT*) — input  
Top row to be scrolled.

**LeftCol** (*USHORT*) — input  
Left column to be scrolled.

**BotRow** (*USHORT*) — input  
Bottom row to be scrolled.

**RightCol** (*USHORT*) — input  
Right column to be scrolled.

**Lines** (*USHORT*) — input  
Number of lines to be inserted at the bottom of the screen area being scrolled. If 0 is specified, no lines are scrolled.

**Cell** (*PBYTE*) — input  
Address of the character attribute(s) pair (2 or 4 bytes) used as a fill character on inserted lines.

**VioHandle** (*HVIO*) — input  
This must be zero unless the caller is a Presentation Manager application, in which case it must be the value returned by VioGetPs.

**rc** (*USHORT*) — return  
Return code descriptions are:

<b>0</b>	<b>NO_ERROR</b>
<b>355</b>	<b>ERROR_VIO_MODE</b>
<b>358</b>	<b>ERROR_VIO_ROW</b>
<b>359</b>	<b>ERROR_VIO_COL</b>
<b>436</b>	<b>ERROR_VIO_INVALID_HANDLE</b>
<b>465</b>	<b>ERROR_VIO_DETACHED</b>

## Remarks

TopRow = 0 and LeftCol = 0 identifies the top left corner of the screen.

If a value greater than the maximum value is specified for TopRow, LeftCol, BotRow, RightCol, or Lines, the maximum value for that parameter is used.

If TopRow and LeftCol = 0 and if BotRow, RightCol, and Lines = 65535 (or -1 in assembler language), the entire screen is filled with the character-attribute pair defined by Cell.

**C Language**

#define INCL\_VIO

```
USHORT rc = VioScrollUp(TopRow, LeftCol, BotRow, RightCol, Lines, Cell,
                        VioHandle);
```

```
USHORT      TopRow;      /* Top row */
USHORT      LeftCol;     /* Left column */
USHORT      BotRow;      /* Bottom row */
USHORT      RightCol;    /* Right column */
USHORT      Lines;       /* Number of lines */
PBYTE       Cell;        /* Fill character */
HVIO        VioHandle;   /* Video handle */

USHORT      rc;          /* return code */
```

**Assembler Language**

```
EXTRN VioScrollUp:FAR
INCL_VIO EQU 1
```

```
PUSH WORD TopRow      ;Top row
PUSH WORD LeftCol     ;Left column
PUSH WORD BotRow      ;Bottom row
PUSH WORD RightCol    ;Right column
PUSH WORD Lines       ;Number of lines
PUSH@ OTHER Cell      ;Fill character
PUSH WORD VioHandle   ;Video handle
CALL VioScrollUp
```

Returns WORD

---

This call releases ownership of (unlocks) the physical display buffer.

<b>VioScrUnLock (VioHandle)</b>
---------------------------------

## Parameters

**VioHandle (HVIO)** — input  
Reserved word of 0s.

**rc (USHORT)** — return  
Return code descriptions are:

<b>0</b>	<b>NO_ERROR</b>
<b>367</b>	<b>ERROR_VIO_UNLOCK</b>
<b>430</b>	<b>ERROR_VIO_ILLEGAL_DURING_POPUP</b>
<b>436</b>	<b>ERROR_VIO_INVALID_HANDLE</b>
<b>465</b>	<b>ERROR_VIO_DETACHED</b>
<b>494</b>	<b>ERROR_VIO_EXTENDED_SG</b>

## Remarks

This call releases the screen lock that is set by VioScrLock. The VioScrUnLock call must be issued by a thread in the same process as the thread that issued VioScrLock.

## Family API Considerations

Some options operate differently in the DOS mode than in the OS/2 mode. Therefore, the following restriction applies to VioScrUnLock when coding in the DOS mode:

The status always indicates the unlock is successful (return code = 0).

## C Language

```
#define INCL_VIO

USHORT rc = VioScrUnLock(VioHandle);

HVIO          VioHandle;    /* Video handle */

USHORT        rc;           /* return code */
```

## Assembler Language

```
EXTRN VioScrUnLock:FAR
INCL_VIO      EQU 1

PUSH WORD    VioHandle    ;Video handle
CALL VioScrUnLock

Returns WORD
```

## VioSetAnsi – Set ANSI On or Off

---

This call activates or deactivates ANSI support.

<b>VioSetAnsi</b> ( <i>Indicator</i> , <i>VioHandle</i> )
---

### Parameters

**Indicator** (*USHORT*) – input

Equals 1 to activate ANSI support or 0 to deactivate ANSI.

**VioHandle** (*HVIO*) – input

This must be zero unless the caller is a Presentation Manager application, in which case it must be the value returned by VioGetPs.

**rc** (*USHORT*) – return

Return code descriptions are:

0	NO_ERROR
355	ERROR_VIO_MODE
421	ERROR_VIO_INVALID_PARMS
430	ERROR_VIO_ILLEGAL_DURING_POPUP
436	ERROR_VIO_INVALID_HANDLE
465	ERROR_VIO_DETACHED

### Remarks

For ANSI support, "ON" is the default.

### C Language

```
#define INCL_VIO
```

```
USHORT rc = VioSetAnsi(Indicator, VioHandle);
```

```
USHORT      Indicator;    /* On/Off indicator */  
HVIO        VioHandle;    /* Video handle */
```

```
USHORT      rc;           /* return code */
```

### Assembler Language

```
EXTRN VioSetAnsi:FAR  
INCL_VIO      EQU 1
```

```
PUSH WORD Indicator    ;On/Off indicator  
PUSH WORD VioHandle    ;Video handle  
CALL VioSetAnsi
```

Returns WORD

# VioSetCp — Set Code Page

This call allows a process to set the code page used to display text data on the screen for the specified handle.

<b>VioSetCp (Reserved, CodePageID, VioHandle)</b>
---

## Parameters

**Reserved (USHORT)** — input  
Reserved word of 0s.

**CodePageID (USHORT)** — input  
The CodePageID must be either 0, -1, -2, or have been specified on the CONFIG.SYS CODEPAGE = statement. A value of 0000 indicates that the code page is to be set to the default ROM code page provided by the hardware. A value of -1 enables the user font codepage if user fonts have previously been set with VioSetFont. A value of -2 disables the user font and re-enables the prepared system codepage or ROM codepage that was active before the user font was enabled.

If the code page ID is not 0, -1, -2, or does not match one of the ID's on the CODEPAGE = statement, an error results. Refer to *IBM Operating System/2 Command Reference* for a complete description of CODEPAGE.

**VioHandle (HVIO)** — input  
This must be zero unless the caller is a Presentation Manager application, in which case it must be the value returned by VioGetPs.

**rc (USHORT)** — return  
Return code descriptions are:

0	NO_ERROR
355	ERROR_VIO_MODE
436	ERROR_VIO_INVALID_HANDLE
465	ERROR_VIO_DETACHED
469	ERROR_VIO_BAD_CP
470	ERROR_VIO_NO_CP
471	ERROR_VIO_NA_CP

## Remarks

VioSetCp can be used to enable and disable the user font code page as well as the prepared system code pages. If a prepared system code page or the ROM code page is specified, any previously set code page is disabled and the specified code page is enabled.

Specifying the special code page of -1 enables the user font code page if user fonts have previously been set. Specifying the special code page of -2 disables the user font code page and re-enables the prepared system code page or ROM code page that was active before the user font code page was enabled.

## PM Considerations

Valid CodePageID values are 0 or one that was specified on the CONFIG.SYS CODEPAGE = statement; -1 and -2 are not valid for PM.

This call can be used to set an EBCDIC code page for Advanced VIO. For a full-screen or Vio-windowed application, this function causes the displayed characters to be reinterpreted immediately in the new code page. For a Presentation Manager application, the characters in the base font are reinterpreted in the new code page only when other events cause the characters to be repainted. This function has no effect on displayed characters that use a font other than the base font.

## VioSetCp – Set Code Page

### C Language

```
#define INCL_VIO

USHORT rc = VioSetCp(Reserved, CodePageID, VioHandle);

USHORT      Reserved;    /* Reserved (must be zero) */
USHORT      CodePageID;  /* CodePage Id */
HVIO        VioHandle;   /* Video handle */

USHORT      rc;          /* return code */
```

### Assembler Language

```
EXTRN VioSetCp:FAR
INCL_VIO EQU 1

PUSH WORD Reserved ;Reserved (must be zero)
PUSH WORD CodePageID ;CodePage Id
PUSH WORD VioHandle ;Video handle
CALL VioSetCp

Returns WORD
```



# VioSetCurPos — Set Cursor Position

FAPI

This call sets the cursor's coordinates on the screen.

**VioSetCurPos (Row, Column, VioHandle)**

## Parameters

**Row** (*USHORT*) — input

New cursor row position, 0 is the top row.

**Column** (*USHORT*) — input

New cursor column position, 0 is the leftmost column.

**VioHandle** (*HVIO*) — input

This must be zero unless the caller is a Presentation Manager application, in which case it must be the value returned by VioGetPs.

**rc** (*USHORT*) — return

Return code descriptions are:

<b>0</b>	<b>NO_ERROR</b>
<b>355</b>	<b>ERROR_VIO_MODE</b>
<b>358</b>	<b>ERROR_VIO_ROW</b>
<b>359</b>	<b>ERROR_VIO_COL</b>
<b>436</b>	<b>ERROR_VIO_INVALID_HANDLE</b>
<b>465</b>	<b>ERROR_VIO_DETACHED</b>

## C Language

```
#define INCL_VIO
```

```
USHORT rc = VioSetCurPos(Row, Column, VioHandle);
```

```
USHORT      Row;          /* Row data */
USHORT      Column;       /* Column data */
HVIO        VioHandle;    /* Video handle */

USHORT      rc;           /* return code */
```

## Assembler Language

```
EXTRN VioSetCurPos:FAR
INCL_VIO EQU 1
```

```
PUSH WORD Row      ;Row data
PUSH WORD Column   ;Column data
PUSH WORD VioHandle ;Video handle
CALL VioSetCurPos
```

Returns WORD

---

This call sets the cursor type.

**VioSetCurType (CursorData, VioHandle)**

## Parameters

**CursorData (PVIOCURSORINFO)** – input

Address of the cursor characteristics structure:

**startline (USHORT)**

Horizontal scan line in the character cell that marks the top line of the cursor. If the character cell has  $n$  scan lines, 0 is the top scan line of the character cell and  $(n - 1)$  is the bottom scan line.

**endline (USHORT)**

Horizontal scan line in the character cell that marks the bottom line of the cursor. Scan lines within a character cell are numbered as defined in startline. The maximum value allowed is 31.

**cursorwidth (USHORT)**

Width of the cursor. In text modes, cursorwidth is the number of columns. The maximum number supported by the OS/2 base video subsystem is 1. In graphics modes, cursorwidth is the number of pels.

A value of 0 specifies the default width. In text modes, this is 1 column. In graphics modes, this is the number of pels equivalent to the width of one character.

**cursorattrib (USHORT)**

A value of  $-1$  denotes a hidden cursor, all other values denote a normal cursor.

**VioHandle (HVIO)** – input

This must be zero unless the caller is a Presentation Manager application, in which case it must be the value returned by VioGetPs.

**rc (USHORT)** – return

Return code descriptions are:

<b>0</b>	<b>NO_ERROR</b>
<b>355</b>	<b>ERROR_VIO_MODE</b>
<b>356</b>	<b>ERROR_VIO_WIDTH</b>
<b>421</b>	<b>ERROR_VIO_INVALID_PARMS</b>
<b>436</b>	<b>ERROR_VIO_INVALID_HANDLE</b>
<b>465</b>	<b>ERROR_VIO_DETACHED</b>

## Remarks

To set CursorStartLine and CursorEndLine independent of the number of scan lines for each character cell, you may specify these parameters as percentages. OS/2 then calculates the physical start and end scan lines, respectively, by multiplying the percentage specified for the parameter by the total number of scan lines in the character cell and rounding to the nearest scan line. Percentages are specified as negative values (or 0) in the range 0 through  $-100$ . Specifying CursorStartLine =  $-90$  and CursorEndLine =  $-100$  requests a cursor that occupies the bottom 10 percent of the character cell.

## PM Considerations

Set the cursor type. The cursor type consists of the cursor start line, end line, width (assumed 0 – one column width) and attribute (normal or hidden).

# VioSetCurType — Set Cursor Type

FAPI

## C Language

```
typedef struct _VIOCURSORINFO { /* vioci */
    USHORT yStart; /*cursor start line */
    USHORT cEnd; /* cursor end line */
    USHORT cx; /* cursor width */
    USHORT attr; /* -1=hidden cursor, any other=normal
                  cursor */
} VIOCURSORINFO;

#define INCL_VIO

USHORT rc = VioSetCurType(CursorData, VioHandle);

PVIOCURSORINFO CursorData; /* Cursor characteristics */
HVIO VioHandle; /* Video handle */

USHORT rc; /* return code */
```

## Assembler Language

```
VIOCURSORINFO struc
    vioci_yStart dw ? ;cursor start line
    vioci_cEnd dw ? ;cursor end line
    vioci_cx dw ? ;cursor width
    vioci_attr dw ? ;-1=hidden cursor, any other=normal cursor
VIOCURSORINFO ends
```

```
EXTRN VioSetCurType:FAR
INCL_VIO EQU 1
```

```
PUSH@ OTHER CursorData ;Cursor characteristics
PUSH WORD VioHandle ;Video handle
CALL VioSetCurType
```

Returns WORD

---

This call downloads a display font. The font being set must be compatible with the current mode.

<b>VioSetFont (RequestBlock, VioHandle)</b>
---

## Parameters

**RequestBlock** (*PVIOFONTINFO*) — input

Address of the font structure containing the request:

**length** (*USHORT*)

Length of structure, including length.

**14** Only valid value.

**reqtype** (*USHORT*)

Request type:

Type	Definition
------	------------

0	Set current RAM font for EGA, VGA, or IBM Personal System/2 Display Adapter.
---	--

**pelcolumns** (*USHORT*)

Pel columns in character cell.

**pelrows** (*USHORT*)

Pel rows in character cell.

**fonttable** (*PVOID*)

Address of the data area containing font table to set.

**tablelength** (*USHORT*)

Length, in bytes, of the caller-supplied data area; must be 256 times the character cell height specified in pelrows.

**VioHandle** (*HVIO*) — input

Reserved word of 0s.

**rc** (*USHORT*) — return

Return code descriptions are:

0	NO_ERROR
355	ERROR_VIO_MODE
421	ERROR_VIO_INVALID_PARMS
436	ERROR_VIO_INVALID_HANDLE
438	ERROR_VIO_INVALID_LENGTH
465	ERROR_VIO_DETACHED
467	ERROR_VIO_FONT
468	ERROR_VIO_USER_FONT
494	ERROR_VIO_EXTENDED_SG

## Remarks

VioSetFont is applicable only for the enhanced graphics adapter, VGA or IBM Personal System/2 Display Adapter.

**Note:** Although graphics mode support is provided in VioSetFont, this support is not provided by the Base Video Handlers provided with OS/2.

When VioSetFont is issued, the current code page is reset. If VioGetCp is subsequently issued, the error code ERROR\_VIO\_USER\_FONT is returned. Return code, ERROR\_VIO\_USER\_FONT represents a warning. It indicates that although the font could not be loaded into the adapter using the current mode, the font was saved as part of a special user font code page for use with a later VioSetMode. Successfully setting a user font sets the special user font code page, just as if a code page of -1 was specified using VioSetCp.

The user font code page consists of the most recent user font of each size that was set by VioSetFont. For example, if two 8x12 fonts and three 8x16 fonts had been set, only two fonts, the most recent of the 8x12 and 8x16 fonts, would be saved.

The special code page is used in the same way as those code pages specified on the CODEPAGE = statement in CONFIG.SYS.

### C Language

```
typedef struct _VIOFONTINFO { /* viofi */
    USHORT cb; /* length of this structure */
    USHORT type; /* request type */
    USHORT cxCell; /* pel columns in character cell */
    USHORT cyCell; /* pel rows in character cell */
    PVOID pbData; /* requested font table (returned) */
    USHORT cbData; /* length of caller supplied data area
                    (in bytes) */
} VIOFONTINFO;

#define INCL_VIO

USHORT rc = VioSetFont(RequestBlock, VioHandle);

PVIOFONTINFO RequestBlock; /* Request block */
HVIO VioHandle; /* Video handle */

USHORT rc; /* return code */
```

### Assembler Language

```
VIOFONTINFO struc
    viofi_cb dw ? ;length of this structure
    viofi_type dw ? ;request type
    viofi_cxCell dw ? ;pel columns in character cell
    viofi_cyCell dw ? ;pel rows in character cell
    viofi_pbData dd ? ;requested font table (returned)
    viofi_cbData dw ? ;length of caller supplied data area (in bytes)
VIOFONTINFO ends

EXTRN VioSetFont:FAR
INCL_VIO EQU 1

PUSH@ OTHER RequestBlock ;Request block
PUSH WORD VioHandle ;Video handle
CALL VioSetFont

Returns WORD
```

---

This call sets the mode of the display.

**VioSetMode (ModeData, VioHandle)**

## Parameters

**ModeData** (*PVIOMODEINFO*) – input

Address of the mode characteristics structure:

**length** (*USHORT*)

Input parameter to VioSetMode. Length specifies the length of the data structure in bytes including Length itself. The minimum structure size required is 3 bytes. OS/2 sets to the first mode (in the list of modes supported by this display configuration) with a data structure matching the mode data specified.

**type** (*UCHAR*)

Mode characteristics bit mask:

Bit	Description
7 – 4	Reserved, set to zero.
3	0 = VGA-compatible modes 0 thru 13H. 1 = Native mode.
2	0 = Enable color burst 1 = Disable color burst.
1	0 = Text mode. 1 = Graphics mode.
0	0 = Monochrome compatible mode. 1 = Other.

**numcolors** (*UCHAR*)

Number of colors defined as a power of 2. This is equivalent to the number of color bits that define the color, for example:

Value	Definition
0	Monochrome modes 7, 7+, and F.
1	2 colors.
2	4 colors.
4	16 colors.
8	256 colors.

**textcols** (*USHORT*)

Number of text columns.

**textrows** (*USHORT*)

Number of text rows.

**pelcols** (*USHORT*)

Horizontal resolution, number of pel columns.

**pelrows** (*USHORT*)

Vertical resolution, number of pel rows.

**Attribute Format** (*UCHAR*)

Identifies the format of the attributes.

# VioSetMode —

## Set Display Mode

FAPi xPM

### Number of Attributes (*UCHAR*)

Identifies the number of attributes in a character cell.

### Buffer Address (*ULONG*)

32-bit physical address of the physical display buffer for this mode.

### Buffer Length (*ULONG*)

Length of the physical display buffer for this mode.

### Full Buffer Size (*ULONG*)

Size of the buffer required for a full save of the physical display buffer for this mode.

### Partial Buffer Size (*ULONG*)

Size of the buffer required for a partial (pop-up) save of the physical display buffer for this mode.

### Extended Data Area Address (*PCH*)

Far address to an extended mode data structure or zero if none. The format of the extended mode data structure is determined by the device driver and is unknown to OS/2.

### VioHandle (*HVIO*) — Input

Reserved word of 0s.

### rc (*USHORT*) — return

Return code descriptions are:

0	NO_ERROR
355	ERROR_VIO_MODE
430	ERROR_VIO_ILLEGAL_DURING_POPUP
436	ERROR_VIO_INVALID_HANDLE
438	ERROR_VIO_INVALID_LENGTH
465	ERROR_VIO_DETACHED
467	ERROR_VIO_FONT
468	ERROR_VIO_USER_FONT
494	ERROR_VIO_EXTENDED_SG

## Remarks

VioSetMode initializes the cursor position and type.

VioSetMode does not clear the screen. To clear the screen, use one of the VioScrollxx calls.

The disable color burst bit in the Type field in the VioSetMode data structure is functional only for the CGA and VGA. This bit causes the color portion of the video signal to be suppressed, producing a black and white mode on composite monitors attached to the CGA. On VGA, the bit causes the color lookup table to be loaded with values that produce shades of gray instead of colors, again producing a black and white mode. For all other combinations of adapters and displays, the setting of this bit is recorded and returned on any subsequent VioGetMode call, but otherwise is ignored.

For text modes in full-screen sessions, the number of rows on the screen is determined by the availability of fonts of the correct size. For any specified mode, the size of the character defined by the font must be (Horizontal Resolution)/(Text Columns) dots wide and (Vertical Resolution)/(Text Rows) dots high. For example, an 8x8 font would support 39 through 43 text rows if the screen resolution were 640x350.

If VioSetState request type 6 has been issued previously to select the target display configuration for VioSetMode, the mode is set on the display configuration selected. If that display configuration does not support the mode specified, an error is returned.

Assuming no target display configuration for VioSetMode is selected, the mode is set on the primary configuration. If the primary configuration does not support the mode specified, the mode is set on the secondary configuration.

The following table shows the VioSetMode parameters required to set all the modes supported by the CGA, EGA, VGA, and PS/2 Display Adapters. The modes native to the 8514/A and other advanced video adapters are set with the Adapter (programming) Interface to these adapters, not VioSetMode.

**Note:** Although graphics mode support is provided in VioSetMode, this support is not provided by the Base Video Handlers provided with OS/2.

**Table 5-1 (Page 1 of 2). Display Mode Attributes Supported by Adapters**

BIOS Mode	Type	Color	Cols	Rows	HRes	VRes	Valid Adapter/Display Combinations [Emulated]
0	5	4	40	25	320	200	[CGA/CD], CGA/Comp, [EGA/CD], [EGA/ECD], VGA/Mono, VGA/Color, VGA/Plasma
0*	5	4	40	25	320	350	[EGA/ECD], VGA/Mono, VGA/Color, VGA/Plasma
0+	5	4	40	25	360	400	VGA/Mono, VGA/Color
0#	5	4	40	25	320	400	VGA/Mono, VGA/Color, VGA/Plasma
1	1	4	40	25	320	200	CGA/CD, CGA/Comp, EGA/CD, EGA/ECD, [VGA/Mono], VGA/Color, [VGA/Plasma]
1*	1	4	40	25	320	350	EGA/ECD, [VGA/Mono], VGA/Color, [VGA/Plasma]
1+	1	4	40	25	360	400	[VGA/Mono], VGA/Color
1#	1	4	40	25	320	400	[VGA/Mono], VGA/Color, [VGA/Plasma]
2	5	4	80	25	640	200	[CGA/CD], CGA/Comp, [EGA/CD], [EGA/ECD], VGA/Mono, VGA/Color, VGA/Plasma
2*	5	4	80	25	640	350	[EGA/ECD], VGA/Mono, VGA/Color, VGA/Plasma
2+	5	4	80	25	720	400	VGA/Mono, VGA/Color
2#	5	4	80	25	640	400	VGA/Mono, VGA/Color, VGA/Plasma
3	1	4	80	25	640	200	CGA/CD, CGA/Comp, EGA/CD, EGA/ECD, [VGA/Mono], VGA/Color, [VGA/Plasma]
3*	1	4	80	25	640	350	EGA/ECD, [VGA/Mono], VGA/Color, [VGA/Plasma]
3+	1	4	80	25	720	400	[VGA/Mono], VGA/Color
3#	1	4	80	25	640	400	[VGA/Mono], VGA/Color, [VGA/Plasma]
7	0	0	80	25	720	350	MPA/MD, EGA/MD, VGA/Mono, VGA/Color
7+	0	0	80	25	720	400	VGA/Mono, VGA/Color
7#	0	0	80	25	640	400	VGA/Mono, VGA/Color, VGA/Plasma
n/a	0	0	80	25	640	350	VGA/Mono, VGA/Color, VGA/Plasma
n/a	1	4	80	30	720	480	[VGA/Mono], VGA/Color
n/a	1	4	80	30	640	480	[VGA/Mono], VGA/Color, [VGA/Plasma]
4	3	2	[40]	[25]	320	200	CGA/CD, CGA/Comp, EGA/CD, EGA/ECD, [VGA/Mono], VGA/Color, [VGA/Plasma]
5	7	2	[40]	[25]	320	200	[CGA/CD], CGA/Comp, [EGA/CD], [EGA/ECD], VGA/Mono, VGA/Color, VGA/Plasma
6	3	1	[80]	[25]	640	200	CGA/CD, CGA/Comp, EGA/CD, EGA/ECD, VGA/Mono, VGA/Color, VGA/Plasma
D	3	4	[40]	[25]	320	200	EGA/CD, EGA/ECD, [VGA/Mono], VGA/Color, [VGA/Plasma]
E	3	4	[80]	[25]	640	200	EGA/CD, EGA/ECD, [VGA/Mono], VGA/Color, [VGA/Plasma]
F	2	0	[80]	[25]	640	350	EGA/MD, VGA/Mono, VGA/Color, VGA/Plasma
10	3	4	[80]	[25]	640	350	EGA/ECD, [VGA/Mono], VGA/Color, [VGA/Plasma]
11	3	1	[80]	[30]	640	480	VGA/Mono, VGA/Color, VGA/Plasma
12	3	4	[80]	[30]	640	480	[VGA/Mono], VGA/Color, [VGA/Plasma]
13	3	8	[40]	[25]	320	200	[VGA/Mono], VGA/Color, [VGA/Plasma]
n/a	11	8	[80]	[30]	640	480	[8514A/Mono], 8514A/Color
n/a	11	4	[80]	[30]	640	480	[8514A/Mono], 8514A/Color
n/a	11	8	[85]	[38]	1024	768	[8514A/HMono], 8514A/HColor



# VioSetMode —

## Set Display Mode

FAPI xPM

Table 5-1 (Page 2 of 2). Display Mode Attributes Supported by Adapters

BIOS Mode	Type	Color	Cols	Rows	HRes	VRes	Valid Adapter/Display Combinations [Emulated]
n/a	11	4	[85]	[38]	1024	768	[8514A/HMono], 8514A/HColor

### Display Adapters:

<b>MPA</b>	Monochrome/Printer Adapter
<b>CGA</b>	Color Graphics Adapter
<b>EGA</b>	Enhanced Graphics Adapter
<b>VGA</b>	Video Graphics Array, PS/2 Display Adapter
<b>8514A</b>	8514/A Display Adapter

### Displays:

<b>MD</b>	5151 Monochrome Display
<b>CD</b>	5153 Color Display
<b>ECD</b>	5154 Enhanced Color Display
<b>Mono</b>	8503 PS/2 Monochrome Display, 8507/8604 Display
<b>HMono</b>	8507/8604 Display
<b>Color</b>	8512/13 PS/2 Color Display, 8514 Display
<b>HColor</b>	8514 Display
<b>Plasma</b>	Plasma Display Panel
<b>Comp</b>	Composite Video Monitor

### Notes:

1. Types 0, 1, and 5 are text modes; types 2, 3, 7, and 11 are graphics modes.
2. For BIOS modes 0, 2, 5, the color burst is disabled on the CGA and VGA.
3. The Personal System/2 Display Adapter 8514/A™<sup>1</sup> has advanced function modes, which are supported through the 8514/A display adapter interface, not the VIO Subsystem. Refer to the *Personal System/2 Display Adapter 8514/A Technical Reference* for details of this support.

**Note:** For text modes in full-screen, the number of rows may differ from the mode table due to the availability of fonts of the correct size as described above.

## PM Considerations

Windowable VIO sessions support only 80-column, color text modes. When VioSetMode is called from a Windowable VIO session, it only verifies that an 80-column text mode was requested, with Text Rows between 1 and 255. The resulting mode, which can be queried using VioGetMode, always has Type = 1, Color = 4, Text Columns = 80, Text Rows = requested Text Rows, Horizontal Resolution = 640, and Vertical Resolution = 16 \* (Text Rows).

---

<sup>1</sup> Trademark of IBM Corporation

## C Language

```
typedef struct _VIOMODEINFO {
    USHORT cb;                /* Length of the entire data structure */
    UCHAR fbType;             /* Bit mask of mode being set */
    UCHAR color;              /* Number of colors (power of 2) */
    USHORT col;               /* Number of text columns */
    USHORT row;               /* Number of text rows */
    USHORT hres;              /* Horizontal resolution */
    USHORT vres;              /* Vertical resolution */
    UCHAR fmt_ID;             /* Attribute format */
    UCHAR attrib;             /* Number of attributes */
    ULONG buf_addr;
    ULONG buf_length;
    ULONG full_length;
    ULONG partial_length;
    PCH ext_data_addr;
} VIOMODEINFO;

typedef VIOMODEINFO far *PVIOMODEINFO;

#define INCL_VIO

USHORT rc = VioSetMode(ModeData, VioHandle);

PVIOMODEINFO ModeData;      /* Mode characteristics */
HVIO VioHandle;             /* Video handle */

USHORT rc;                  /* return code */
```

## Assembler Language

```
VIOMODEINFO struc
    viomi_cb          dw ? ;Length of the entire data structure
    viomi_fbType      db ? ;Bit mask of mode being set
    viomi_color       db ? ;Number of colors (power of 2)
    viomi_col         dw ? ;Number of text columns
    viomi_row         dw ? ;Number of text rows
    viomi_hres        dw ? ;Horizontal resolution
    viomi_vres        dw ? ;Vertical resolution
    viomi_fmt_ID      db ? ;Attribute format
    viomi_attrib       db ? ;Number of attributes
    viomi_buf_addr    dd ? ;
    viomi_buf_length  dd ? ;
    viomi_full_length dd ? ;
    viomi_partial_length dd ? ;
    viomi_ext_data_addr dd ? ;
VIOMODEINFO ends

EXTRN VioSetMode:FAR
INCL_VIO EQU 1

PUSH@ OTHER ModeData ;Mode characteristics
PUSH WORD VioHandle ;Video handle
CALL VioSetMode

Returns WORD
```

# VioSetState —

## Set Video State

FAPI xWPM

This call performs one of the following functions; set palette registers, sets the overscan (border) color, set the blink/background intensity switch, set color registers, set the underline location, or set the target VioSetMode display configuration.

**VioSetState (RequestBlock, VioHandle)**

### Parameters

**RequestBlock (PVOID)** — input

Address of the video state structures consisting of six different structures depending on the request type:

Type	Definition
0	Set palette registers
1	Set overscan (border) color
2	Set blink/background intensity switch
3	Set color registers
4	Reserved
5	Set underline location
6	Set target VioSetMode display configuration
7	Reserved

The six structures, depending on request type, are:

#### VIOPALSTATE

Applies to EGA, VGA, or IBM Personal System/2 Display Adapter.

**length (USHORT)** — input

Length of structure, including length.

**38** Maximum valid value.

**reqtype (USHORT)** — input

Request type 0 for palette registers.

**palette (USHORT)** — input

First palette register in the palette register sequence; must be specified in the range 0 through 15. The palette registers are returned in sequential order. The number returned is based upon length.

**color (USHORT\*(length-6)/2)** — input

Color value for each palette register. The maximum number of entries in the color value array is 16.

#### VIOOVERSCAN

Applies to CGA, VGA, or IBM Personal System/2 Display Adapter.

**length (USHORT)** — input

Length of structure, including length.

**6** Only valid value.

**reqtype (USHORT)** — input

Request type 1 for overscan (border) color.

**color (USHORT)** — input

Color value.

#### VIOINTENSITY

Applies to CGA, EGA, MCGA, VGA, or IBM Personal System/2 Display Adapter.

**length (USHORT)** — input

Length of structure, including length.

**6** Only valid value.

**reqtype** (*USHORT*) — input  
Request type 2 for blink/background intensity switch.

**switch** (*USHORT*) — input  
Switch set as:

Value	Definition
0	Blinking foreground colors enabled.
1	High intensity background colors enabled.

## VIOCOLORREG

Applies to VGA, or IBM Personal System/2 Display Adapter.

**length** (*USHORT*) — input  
Length of structure, including length.

**12** Only valid value.

**type** (*USHORT*) — input  
Request type 3 for color registers.

**first color** (*USHORT*) — input  
First color register to set in the color register sequence; must be specified in the range 0 through 255. The color registers are set in sequential order.

**number color** (*USHORT*) — input  
Number of color registers to set; must be specified in the range 1 through 256.

**dataarea** (*PCH*) — input  
Far address of a data area containing one three-byte entry for each color register to be set. The format of each entry is as follows:

<b>Byte 1</b>	Red value
<b>Byte 2</b>	Green value
<b>Byte 3</b>	Blue value.

## VIOSETULINELOC

Applies to EGA, VGA, or IBM Personal System/2 Display Adapter.

**length** (*USHORT*) — input  
Length of structure, including length.

**6** Only valid value.

**type** (*USHORT*) — input  
Request type 5 to set the scan line for underlining. Underlining is enabled only when the foreground color is 1 or 9.

**scanline** (*USHORT*) — input  
Scan line minus 1. Values of 0 through 31 are acceptable. A value of 32 means underlining is disabled.

## VIOSETTARGET

**length** (*USHORT*) — input  
Length of structure, including length.

**6** Only valid value.

**type** (*USHORT*) — input  
Request type 6 to set display configuration to be the target of the next VioSetMode.

**select** (*USHORT*) — input  
Configuration:

Value	Definition
0	Default selection algorithm. See VioSetMode.
1	Primary
2	Secondary.

# VioSetState — Set Video State

FAP1 xWPM

**VioHandle** (*HVIO*) — input  
Reserved word of 0s.

**rc** (*USHORT*) — return  
Return code descriptions are:

<b>0</b>	<b>NO_ERROR</b>
<b>355</b>	<b>ERROR_VIO_MODE</b>
<b>421</b>	<b>ERROR_VIO_INVALID_PARMS</b>
<b>436</b>	<b>ERROR_VIO_INVALID_HANDLE</b>
<b>438</b>	<b>ERROR_VIO_INVALID_LENGTH</b>
<b>465</b>	<b>ERROR_VIO_DETACHED</b>
<b>494</b>	<b>ERROR_VIO_EXTENDED_SG</b>

## Family API Considerations

Request type = 6, Set Target VioSetMode Display Configuration, and request type = 5, Set Underline Location, are not supported in the family API.

**C Language**

```

typedef struct _VIOPALSTATE {
    USHORT cb;                /* Length of this structure in bytes */
    USHORT type;              /* Request type=0 get palette registers */
    USHORT iFirst;            /* First palette register to return */
    USHORT acolor[1];         /* Color value palette register */
}VIOPALSTATE;
typedef VIOPALSTATE far *PVIOPALSTATE;

typedef struct _VIOOVERSCAN {
    USHORT cb;                /* Length of this structure */
    USHORT type;              /* Request type=1 get overscan (border) color */
    USHORT color;             /* Color value */
}VIOOVERSCAN;
typedef VIOOVERSCAN far *PVIOOVERSCAN;

typedef struct _VIOINTENSITY {
    USHORT cb;                /* Length of this structure */
    USHORT type;              /* Request type=2 get blink/background
                             intensity switch */
    USHORT fs;                /* Value of blink/background switch */
}VIOINTENSITY;
typedef VIOINTENSITY far *PVIOINTENSITY;

typedef struct _VIOCOLORREG { /* viocreg */
    USHORT cb;
    USHORT type;
    USHORT firstcolorreg;
    USHORT numcolorregs;
    PCH colorregaddr;
}VIOCOLORREG;
typedef VIOCOLORREG far *PVIOCOLORREG;

typedef struct _VIOSETLINELOC { /* viouline */
    USHORT cb;
    USHORT type;
    USHORT scanline;
}VIOSETLINELOC;
typedef VIOSETLINELOC far *PVIOSETLINELOC;

typedef struct _VIOSETTARGET { /* viosett */
    USHORT cb;
    USHORT type;
    USHORT defaultalgorithm;
}VIOSETTARGET;
typedef VIOSETTARGET far *PVIOSETTARGET;

#define INCL_VIO

USHORT rc = VioSetState(RequestBlock, VioHandle);

PVOID      RequestBlock; /* Request block */
HVIO       VioHandle;    /* Video handle */

USHORT     rc;            /* return code */

```

# VioSetState — Set Video State

FAPI xWPM

## Assembler Language

```
VIOPALSTATE struc
    viopal_cb          dw ? ;Length of this structure in bytes
    viopal_type        dw ? ;Request type=0 get palette registers
    viopal_iFirst      dw ? ;First palette register to return
    viopal_acolor      dw 1 dup (?) ;Color value palette register
VIOPALSTATE ends

VIOOVERSCAN struc
    vioos_cb          dw ? ;Length of this structure
    vioos_type        dw ? ;Request type=1 get overscan (border) color
    vioos_color        dw ? ;Color value
VIOOVERSCAN ends

VIOINTENSITY struc
    vioint_cb         dw ? ;Length of this structure
    vioint_type        dw ? ;Request type=2 get blink/background intensity switch
    vioint_fs         dw ? ;Value of blink/background switch
VIOINTENSITY ends

VIOCOLORREG struc
    viocreg_cb        dw ? ;
    viocreg_type       dw ? ;
    viocreg_firstcolorreg dw ? ;
    viocreg_numcolorregs dw ? ;
    viocreg_colorregaddr dd ? ;
VIOCOLORREG ends

VIOSETLINELOC struc
    viouline_cb       dw ? ;
    viouline_type      dw ? ;
    viouline_scanline dw ? ;
VIOSETLINELOC ends

VIOSETTARGET struc
    viosett_cb        dw ? ;
    viosett_type       dw ? ;
    viosett_defaultalgorithm dw ? ;
VIOSETTARGET ends

EXTRN VioSetState:FAR
INCL_VIO EQU 1

PUSH@ OTHER RequestBlock ;Request block
PUSH WORD VioHandle ;Video handle
CALL VioSetState

Returns WORD
```

# VioShowBuf – Display Logical Buffer

This call updates the physical display buffer with the logical video buffer (LVB).

**VioShowBuf (Offset, Length, VioHandle)**

## Parameters

**Offset** (*USHORT*) – input

Starting offset within the logical video buffer at which the update to the screen is to start.

**Length** (*USHORT*) – input

Length of the area to be updated to the screen.

**VioHandle** (*HVIO*) – input

This must be zero unless the caller is a Presentation Manager application, in which case it must be the value returned by VioGetPs.

**rc** (*USHORT*) – return

Return code descriptions are:

<b>0</b>	<b>NO_ERROR</b>
<b>355</b>	<b>ERROR_VIO_MODE</b>
<b>430</b>	<b>ERROR_VIO_ILLEGAL_DURING_POPUP</b>
<b>436</b>	<b>ERROR_VIO_INVALID_HANDLE</b>
<b>465</b>	<b>ERROR_VIO_DETACHED</b>

## Remarks

VioShowBuf is ignored unless it is issued by a process that has previously called VioGetBuf and that is currently executing in the foreground.

## PM Considerations

This function updates the display with the Advanced VIO presentation space.

## C Language

```
#define INCL_VIO

USHORT rc = VioShowBuf(Offset, Length, VioHandle);

USHORT      Offset;      /* Offset into LVB */
USHORT      Length;      /* Length */
HVIO        VioHandle;    /* Video handle */

USHORT      rc;          /* return code */
```

## Assembler Language

```
EXTRN VioShowBuf:FAR
INCL_VIO EQU 1

PUSH WORD Offset      ;Offset into LVB
PUSH WORD Length      ;Length
PUSH WORD VioHandle    ;Video handle
CALL VioShowBuf

Returns WORD
```



# VioWrtCellStr – Write Char/Attr String

FAPI

This call writes a string of character-attribute pairs (cells) to the display.

**VioWrtCellStr (CellStr, Length, Row, Column, VioHandle)**

## Parameters

**CellStr** (*PCH*) – input

Address of the string of character-attribute(s) cells to be written.

**Length** (*USHORT*) – input

Length, in bytes, of the string to be written. Each character-attribute(s) cell is 2 or 4 bytes.

**Row** (*USHORT*) – input

Starting cursor row.

**Column** (*USHORT*) – input

Starting cursor column.

**VioHandle** (*HVIO*) – input

This must be zero unless the caller is a Presentation Manager application, in which case it must be the value returned by VioGetPs.

**rc** (*USHORT*) – return

Return code descriptions are:

<b>0</b>	<b>NO_ERROR</b>
<b>355</b>	<b>ERROR_VIO_MODE</b>
<b>358</b>	<b>ERROR_VIO_ROW</b>
<b>359</b>	<b>ERROR_VIO_COL</b>
<b>436</b>	<b>ERROR_VIO_INVALID_HANDLE</b>
<b>465</b>	<b>ERROR_VIO_DETACHED</b>

## Remarks

If a string write gets to the end of the line and is not complete, the string write continues at the beginning of the next line. If the write gets to the end of the screen, the write terminates.

## PM Considerations

Write a character-attribute string to the Advanced VIO presentation space. The caller must specify the starting location on the presentation space where the string is to be written.

## C Language

```
#define INCL_VIO

USHORT rc = VioWrtCellStr(CellStr, Length, Row, Column, VioHandle);

PCH      CellStr;      /* String to be written */
USHORT   Length;       /* Length of string */
USHORT   Row;          /* Starting row position for output */
USHORT   Column;       /* Starting column position for output */
HVIO     VioHandle;    /* Video handle */

USHORT   rc;           /* return code */
```

**Assembler Language**

```
EXTRN VioWrtCellStr:FAR
INCL_VIO EQU 1

PUSH@ OTHER CellStr      ;String to be written
PUSH WORD Length        ;Length of string
PUSH WORD Row           ;Starting row position for output
PUSH WORD Column        ;Starting column position for output
PUSH WORD VioHandle     ;Video handle
CALL VioWrtCellStr
```

Returns WORD

# VioWrtCharStr — Write Character String

FAPI

This call writes a character string to the display.

**VioWrtCharStr (CharStr, Length, Row, Column, VioHandle)**

## Parameters

**CharStr (PCH)** — input

Address of the character string to be written.

**Length (USHORT)** — input

Length, in bytes, of the character string.

**Row (USHORT)** — input

Starting cursor row.

**Column (USHORT)** — input

Starting cursor column.

**VioHandle (HVIO)** — input

This must be zero unless the caller is a Presentation Manager application, in which case it must be the value returned by VioGetPs.

**rc (USHORT)** — return

Return code descriptions are:

<b>0</b>	<b>NO_ERROR</b>
<b>355</b>	<b>ERROR_VIO_MODE</b>
<b>358</b>	<b>ERROR_VIO_ROW</b>
<b>359</b>	<b>ERROR_VIO_COL</b>
<b>436</b>	<b>ERROR_VIO_INVALID_HANDLE</b>
<b>465</b>	<b>ERROR_VIO_DETACHED</b>

## Remarks

If a string write gets to the end of the line and is not complete, the string write continues at the beginning of the next line. If the write gets to the end of the screen, the write terminates.

## PM Considerations

Write a character string to the Advanced VIO presentation space. The caller must specify the starting location on the presentation space where the string is to be written.

## C Language

```
#define INCL_VIO
```

```
USHORT rc = VioWrtCharStr(CharStr, Length, Row, Column, VioHandle);
```

```
PCH          CharStr;      /* String to be written */
USHORT       Length;       /* Length of character string */
USHORT       Row;          /* Starting row position for output */
USHORT       Column;       /* Starting column position for output */
HVIO         VioHandle;    /* Video handle */

USHORT       rc;           /* return code */
```

**Assembler Language**

```
EXTRN VioWrtCharStr:FAR
INCL_VIO      EQU 1

PUSH@ OTHER CharStr      ;String to be written
PUSH WORD Length        ;Length of character string
PUSH WORD Row           ;Starting row position for output
PUSH WORD Column        ;Starting column position for output
PUSH WORD VioHandle     ;Video handle
CALL VioWrtCharStr
```

Returns WORD

# VioWrtCharStrAtt — Write Char String with Attr

FAPI

This call writes a character string with repeated attribute to the display.

**VioWrtCharStrAtt (CharStr, Length, Row, Column, Attr, VioHandle)**

## Parameters

**CharStr (PCH)** — input

Address of the character string to be written.

**Length (USHORT)** — input

Length, in bytes, of the character string.

**Row (USHORT)** — input

Starting cursor row.

**Column (USHORT)** — input

Starting cursor column.

**Attr (PBYTE)** — input

Address of the attribute(s) (1 or 3 bytes) to be used in the display buffer for each character of the string written.

**VioHandle (HVIO)** — input

This must be zero unless the caller is a Presentation Manager application, in which case it must be the value returned by VioGetPs.

**rc (USHORT)** — return

Return code descriptions are:

0	NO_ERROR
355	ERROR_VIO_MODE
358	ERROR_VIO_ROW
359	ERROR_VIO_COL
436	ERROR_VIO_INVALID_HANDLE
465	ERROR_VIO_DETACHED

## Remarks

If a string write gets to the end of the line and is not complete, the string write continues at the beginning of the next line. If the write gets to the end of the screen, the write terminates.

## PM Considerations

Write a character string with a repeated attribute string to the Advanced VIO presentation space. The caller must specify the starting location on the presentation space where the string is to be written.

## C Language

```
#define INCL_VIO
```

```
USHORT rc = VioWrtCharStrAtt(CharStr, Length, Row, Column, Attr, VioHandle);
```

PCH	CharStr;	/* String to be written */
USHORT	Length;	/* Length of string */
USHORT	Row;	/* Starting row position for output */
USHORT	Column;	/* Starting column position for output */
PBYTE	Attr;	/* Attribute to be replicated */
HVIO	VioHandle;	/* Video handle */
USHORT	rc;	/* return code */

**Assembler Language**

```
EXTRN VioWrtCharStrAtt:FAR
INCL_VIO      EQU 1

PUSH@ OTHER CharStr      ;String to be written
PUSH WORD Length        ;Length of string
PUSH WORD Row           ;Starting row position for output
PUSH WORD Column        ;Starting column position for output
PUSH@ OTHER Attr        ;Attribute to be replicated
PUSH WORD VioHandle     ;Video handle
CALL VioWrtCharStrAtt
```

Returns WORD

This call writes an attribute to the display a specified number of times.

**VioWrtNAttr (Attr, Times, Row, Column, VioHandle)**

## Parameters

**Attr** (*PBYTE*) – input

Address of the attribute(s) (1 or 3 bytes) to be written.

**Times** (*USHORT*) – input

Number of times to write the attribute.

**Row** (*USHORT*) – input

Starting cursor row.

**Column** (*USHORT*) – input

Starting cursor column.

**VioHandle** (*HVIO*) – input

This must be zero unless the caller is a Presentation Manager application, in which case it must be the value returned by VioGetPs.

**rc** (*USHORT*) – return

Return code descriptions are:

<b>0</b>	<b>NO_ERROR</b>
<b>355</b>	<b>ERROR_VIO_MODE</b>
<b>358</b>	<b>ERROR_VIO_ROW</b>
<b>359</b>	<b>ERROR_VIO_COL</b>
<b>436</b>	<b>ERROR_VIO_INVALID_HANDLE</b>
<b>465</b>	<b>ERROR_VIO_DETACHED</b>

## Remarks

If a repeated write gets to the end of the line and is not complete, the write continues at the beginning of the next line. If the write gets to the end of the screen, the write terminates.

## PM Considerations

Write an attribute code to the Advanced VIO presentation space a specified number of times. The caller must specify the starting location on the presentation space where the string is to be written.

## C Language

```
#define INCL_VIO
```

```
USHORT rc = VioWrtNAttr(Attr, Times, Row, Column, VioHandle);
```

```
PBYTE      Attr;          /* Attribute to be written */
USHORT      Times;         /* Repeat count */
USHORT      Row;           /* Starting row position for output */
USHORT      Column;        /* Starting column position for output */
HVIO        VioHandle;     /* Video handle */

USHORT      rc;            /* return code */
```

**Assembler Language**

```
EXTRN  VioWrtNAttr:FAR
INCL_VIO      EQU 1

PUSH@ OTHER  Attr      ;Attribute to be written
PUSH  WORD   Times     ;Repeat count
PUSH  WORD   Row       ;Starting row position for output
PUSH  WORD   Column    ;Starting column position for output
PUSH  WORD   VioHandle  ;Video handle
CALL  VioWrtNAttr

Returns WORD
```



---

This call writes a cell (character-attribute pair) to the display a specified number of times.

**VioWrtNCell (Cell, Times, Row, Column, VioHandle)**

## Parameters

**Cell** (*PBYTE*) — input

Address of the character-attribute(s) cell (2 or 4 bytes) to be written.

**Times** (*USHORT*) — input

Number of times to write the cell.

**Row** (*USHORT*) — input

Starting cursor row.

**Column** (*USHORT*) — input

Starting cursor column.

**VioHandle** (*HVIO*) — input

This must be zero unless the caller is a Presentation Manager application, in which case it must be the value returned by VioGetPs.

**rc** (*USHORT*) — return

Return code descriptions are:

<b>0</b>	<b>NO_ERROR</b>
<b>355</b>	<b>ERROR_VIO_MODE</b>
<b>358</b>	<b>ERROR_VIO_ROW</b>
<b>359</b>	<b>ERROR_VIO_COL</b>
<b>436</b>	<b>ERROR_VIO_INVALID_HANDLE</b>
<b>465</b>	<b>ERROR_VIO_DETACHED</b>

## Remarks

If a repeated write gets to the end of the line and is not complete, the write continues at the beginning of the next line. If the write gets to the end of the screen, the write terminates.

## PM Considerations

Write a cell (character-attribute) to the Advanced VIO presentation space a specified number of times. The caller must specify the starting location on the presentation space where the string is to be written.

## C Language

```
#define INCL_VIO
```

```
USHORT rc = VioWrtNCell(Cell, Times, Row, Column, VioHandle);
```

```
PBYTE      Cell;          /* Cell to be written */
USHORT      Times;         /* Repeat count */
USHORT      Row;           /* Starting row position for output */
USHORT      Column;        /* Starting column position for output */
HVIO        VioHandle;     /* Video handle */

USHORT      rc;            /* return code */
```

**Assembler Language**

```
EXTRN VioWrtNCell:FAR
INCL_VIO      EQU 1

PUSH@ OTHER   Cell           ;Cell to be written
PUSH  WORD    Times         ;Repeat count
PUSH  WORD    Row           ;Starting row position for output
PUSH  WORD    Column        ;Starting column position for output
PUSH  WORD    VioHandle     ;Video handle
CALL  VioWrtNCell

Returns WORD
```

# VioWrtNChar — Write N Characters

FAPI

VioWrtNChar writes a character to the display a specified number of times.

**VioWrtNChar (Char, Times, Row, Column, VioHandle)**

## Parameters

**Char** (*PCH*) — input

Address of the character to be written.

**Times** (*USHORT*) — input

Number of times to write the character.

**Row** (*USHORT*) — input

Starting cursor row.

**Column** (*USHORT*) — input

Starting cursor column.

**VioHandle** (*HVIO*) — input

This must be zero unless the caller is a Presentation Manager application, in which case it must be the value returned by VioGetPs.

**rc** (*USHORT*) — return

Return code descriptions are:

<b>0</b>	<b>NO_ERROR</b>
<b>355</b>	<b>ERROR_VIO_MODE</b>
<b>358</b>	<b>ERROR_VIO_ROW</b>
<b>359</b>	<b>ERROR_VIO_COL</b>
<b>436</b>	<b>ERROR_VIO_INVALID_HANDLE</b>
<b>465</b>	<b>ERROR_VIO_DETACHED</b>

## Remarks

If a repeated write gets to the end of the line and is not complete, the write continues at the beginning of the next line. If the write gets to the end of the screen, the write terminates.

## PM Considerations

Write a character to the Advanced VIO presentation space a number of times. The caller must specify the starting location on the presentation space where the string is to be written.

## C Language

```
#define INCL_VIO
```

```
USHORT rc = VioWrtNChar(Char, Times, Row, Column, VioHandle);
```

```
PCH          Char;          /* Character to be written */
USHORT       Times;         /* Repeat count */
USHORT       Row;           /* Starting row position for output */
USHORT       Column;        /* Starting column position for output */
HVIO         VioHandle;     /* Video handle */

USHORT       rc;            /* return code */
```

**Assembler Language**

```
EXTRN  VioWrtNChar:FAR
INCL_VIO      EQU 1

PUSH@ OTHER  Char      ;Character to be written
PUSH  WORD   Times     ;Repeat count
PUSH  WORD   Row       ;Starting row position for output
PUSH  WORD   Column    ;Starting column position for output
PUSH  WORD   VioHandle  ;Video handle
CALL  VioWrtNChar

Returns WORD
```

This call writes a character string to the display starting at the current cursor position. At the completion of the write, the cursor is positioned at the first position beyond the end of the string.

**VioWrtTTY (CharStr, Length, VioHandle)**

## Parameters

**CharStr (PCH)** – input

Address of the string to be written.

**Length (USHORT)** – input

Length of the character string in bytes.

**VioHandle (HVIO)** – input

This must be zero unless the caller is a Presentation Manager application, in which case it must be the value returned by VioGetPs.

**rc (USHORT)** – return

Return code descriptions are:

<b>0</b>	<b>NO_ERROR</b>
<b>355</b>	<b>ERROR_VIO_MODE</b>
<b>436</b>	<b>ERROR_VIO_INVALID_HANDLE</b>
<b>465</b>	<b>ERROR_VIO_DETACHED</b>

## Remarks

If a string write gets to the end of the line and is not complete, the string write continues at the beginning of the next line. If the write gets to the end of the screen, the screen is scrolled, and the write continues until completed.

The characters carriage return, line feed, backspace, tab, and bell are treated as commands rather than printable characters. Backspace is a non-destructive backspace. Tabs are expanded to provide standard 8-byte-wide fields. VioWrtTTY is the only video call affected by Ctrl-PrtSc and ANSI.

Characters are written using the current attribute defined by ANSI or the default value 7.

VioWrtTTY is supported in graphics mode to process ANSI sequences. This allows the application to enter and exit a graphics mode.

## PM Considerations

Write a character string from the current cursor position in TTY mode to the Advanced VIO presentation space. The cursor is positioned after the last character written at the end of the write.

## C Language

```
#define INCL_VIO

USHORT rc = VioWrtTTY(CharStr, Length, VioHandle);

PCH      CharStr;      /* String to be written */
USHORT   Length;       /* Length of string */
HVIO     VioHandle;    /* Video handle */

USHORT   rc;           /* return code */
```

**Assembler Language**

```
EXTRN  VioWrtTTY:FAR
INCL_VIO      EQU 1

PUSH@ OTHER  CharStr      ;String to be written
PUSH  WORD   Length      ;Length of string
PUSH  WORD   VioHandle    ;Video handle
CALL  VioWrtTTY

Returns WORD
```



---

## Appendix A. Errors Returned from Base OS/2 Calls

### Numeric Order

Error Number and Name	Description
0 NO_ERROR	No error occurred.
1 ERROR_INVALID_FUNCTION	Invalid function number.
2 ERROR_FILE_NOT_FOUND	File not found.
3 ERROR_PATH_NOT_FOUND	Path not found.
4 ERROR_TOO_MANY_OPEN_FILES	Too many open files (no handles left).
5 ERROR_ACCESS_DENIED	Access denied.
6 ERROR_INVALID_HANDLE	Invalid handle.
7 ERROR_ARENA_TRASHED	Memory control blocks destroyed.
8 ERROR_NOT_ENOUGH_MEMORY	Insufficient memory.
9 ERROR_INVALID_BLOCK	Invalid memory-block address.
10 ERROR_BAD_ENVIRONMENT	Invalid environment.
11 ERROR_BAD_FORMAT	Invalid format.
12 ERROR_INVALID_ACCESS	Invalid access code.
13 ERROR_INVALID_DATA	Invalid data.
14	Reserved.
15 ERROR_INVALID_DRIVE	Invalid drive specified.
16 ERROR_CURRENT_DIRECTORY	Attempting to remove current directory.
17 ERROR_NOT_SAME_DEVICE	Not same device.
18 ERROR_NO_MORE_FILES	No more files.
19 ERROR_WRITE_PROTECT	Attempt to write on write-protected diskette.
20 ERROR_BAD_UNIT	Unknown unit.
21 ERROR_NOT_READY	Drive not ready.
22 ERROR_BAD_COMMAND	Unknown command.
23 ERROR_CRC	Data error (CRC).
24 ERROR_BAD_LENGTH	Bad request structure length.
25 ERROR_SEEK	Seek error.
26 ERROR_NOT_DOS_DISK	Unknown media type.
27 ERROR_SECTOR_NOT_FOUND	Sector not found.
28 ERROR_OUT_OF_PAPER	Printer out of paper.
29 ERROR_WRITE_FAULT	Write fault.
30 ERROR_READ_FAULT	Read fault.
31 ERROR_GEN_FAILURE	General failure.
32 ERROR_SHARING_VIOLATION	Sharing violation.
33 ERROR_LOCK_VIOLATION	Lock violation.
34 ERROR_WRONG_DISK	Invalid disk change.
35 ERROR_FCB_UNAVAILABLE	FCB unavailable.
36 ERROR_SHARING_BUFFER_EXCEEDED	Sharing buffer overflow.
37–49	Reserved.
50 ERROR_NOT_SUPPORTED	Network request not supported.
65	Access denied.
73–79	Reserved.
80 ERROR_FILE_EXISTS	File exists.
81 ERROR_DUP_FCB	Reserved.
82 ERROR_CANNOT_MAKE	Cannot make directory entry.
83 ERROR_FAIL_I24	Fail on INT 24.
84 ERROR_OUT_OF_STRUCTURES	Too many redirections.
85 ERROR_ALREADY_ASSIGNED	Duplicate redirection.
86 ERROR_INVALID_PASSWORD	Invalid password.
87 ERROR_INVALID_PARAMETER	Invalid parameter.
88 ERROR_NET_WRITE_FAULT	Network device fault.
89 ERROR_NO_PROC_SLOTS	No process slots available.
90 ERROR_NOT_FROZEN	System error.
91 ERR_TSTOVFL	Timer service table overflow.
92 ERR_TSTDUP	Timer service table duplicate.
93 ERROR_NO_ITEMS	No items to work on.
95 ERROR_INTERRUPT	Interrupted system call.



99 ERROR_DEVICE_IN_USE	Device in use.
100 ERROR_TOO_MANY_SEMAPHORES	User/system open semaphore limit exceeded.
101 ERROR_EXCL_SEM_ALREADY_OWNED	Exclusive semaphore already owned.
102 ERROR_SEM_IS_SET	DosCloseSem found semaphore set.
103 ERROR_TOO_MANY_SEM_REQUESTS	Too many exclusive semaphore requests.
104 ERROR_INVALID_AT_INTERRUPT_TIME	Operation invalid at interrupt time.
105 ERROR_SEM_OWNER_DIED	Previous semaphore owner terminated without freeing semaphore.
106 ERROR_SEM_USER_LIMIT	Semaphore limit exceeded.
107 ERROR_DISK_CHANGE	Insert drive B disk into drive A.
108 ERROR_DRIVE_LOCKED	Drive locked by another process.
109 ERROR_BROKEN_PIPE	Write on pipe with no reader.
110 ERROR_OPEN_FAILED	Open/create failed due to explicit fail command.
111 ERROR_BUFFER_OVERFLOW	Buffer passed to system call too small to hold return data.
112 ERROR_DISK_FULL	Not enough space on the disk.
113 ERROR_NO_MORE_SEARCH_HANDLES	Cannot allocate another search structure and handle.
114 ERROR_INVALID_TARGET_HANDLE	Target handle in DosDupHandle invalid.
115 ERROR_PROTECTION_VIOLATION	Bad user virtual address.
116 ERROR_VIOKBD_REQUEST	Error on display write or keyboard read.
117 ERROR_INVALID_CATEGORY	Category for DevIOCtl not defined.
118 ERROR_INVALID_VERIFY_SWITCH	Invalid value passed for verify flag.
119 ERROR_BAD_DRIVER_LEVEL	Level four driver not found.
120 ERROR_CALL_NOT_IMPLEMENTED	Invalid function called.
121 ERROR_SEM_TIMEOUT	Time out occurred from semaphore API function.
122 ERROR_INSUFFICIENT_BUFFER	Data buffer too small.
123 ERROR_INVALID_NAME	Illegal character or bad file-system name.
124 ERROR_INVALID_LEVEL	Non-implemented level for information retrieval or setting.
125 ERROR_NO_VOLUME_LABEL	No volume label found with DosQFInfo command.
126 ERROR_MOD_NOT_FOUND	Module handle not found with getprocaddr, getmodhandle.
127 ERROR_PROC_NOT_FOUND	Procedure address not found with getprocaddr.
128 ERROR_WAIT_NO_CHILDREN	DosCwait finds no children.
129 ERROR_CHILD_NOT_COMPLETE	DosCwait children not terminated.
130 ERROR_DIRECT_ACCESS_HANDLE	Handle operation invalid for direct disk-access handles.
131 ERROR_NEGATIVE_SEEK	Attempting seek to negative offset.
132 ERROR_SEEK_ON_DEVICE	Application trying to seek on device or pipe.
133 ERROR_IS_JOIN_TARGET	Drive has previously joined drives.
134 ERROR_IS_JOINED	Drive is already joined.
135 ERROR_IS_SUBSTED	Drive is already substituted.
136 ERROR_NOT_JOINED	Cannot delete drive that is not joined.
137 ERROR_NOT_SUBSTED	Cannot delete drive that is not substituted.
138 ERROR_JOIN_TO_JOIN	Cannot join to a joined drive.
139 ERROR_SUBST_TO_SUBST	Cannot substitute to a substituted drive.
140 ERROR_JOIN_TO_SUBST	Cannot join to a substituted drive.
141 ERROR_SUBST_TO_JOIN	Cannot substitute to a joined drive.
142 ERROR_BUSY_DRIVE	Specified drive is busy.
143 ERROR_SAME_DRIVE	Cannot join or substitute a drive to a directory on the same drive.
144 ERROR_DIR_NOT_ROOT	Directory must be a subdirectory of the root.
145 ERROR_DIR_NOT_EMPTY	Directory must be empty to use join command.
146 ERROR_IS_SUBST_PATH	Path specified is being used in a substitute.
147 ERROR_IS_JOIN_PATH	Path specified is being used in join.
148 ERROR_PATH_BUSY	Path specified is being used by another process.
149 ERROR_IS_SUBST_TARGET	Cannot join or substitute drive having directory that is target of a previous substitute.
150 ERROR_SYSTEM_TRACE	System trace error.
151 ERROR_INVALID_EVENT_COUNT	DosMuxSemWait errors.
152 ERROR_TOO_MANY_MUXWAITERS	System limit of 100 entries reached.
153 ERROR_INVALID_LIST_FORMAT	Invalid list format.
154 ERROR_LABEL_TOO_LONG	Volume label too big.
155 ERROR_TOO_MANY_TCBS	Cannot create another TCB.
156 ERROR_SIGNAL_REFUSED	Signal refused.
157 ERROR_DISCARDED	Segment is discarded.
158 ERROR_NOT_LOCKED	Segment not locked.

159 ERROR_BAD_THREADID_ADDR	Bad thread-identity address.
160 ERROR_BAD_ARGUMENTS	Bad environment pointer.
161 ERROR_BAD_PATHNAME	Bad path name passed to exec.
162 ERROR_SIGNAL_PENDING	Signal already pending.
163 ERROR_UNCERTAIN_MEDIA	ERROR_I24 mapping.
164 ERROR_MAX_THRDS_REACHED	No more process slots.
165 ERROR_MONITORS_NOT_SUPPORTED	ERROR_I24 mapping.
166 ERROR_UNC_DRIVER_NOT_INSTALLED	Default redir return code
167 ERROR_LOCK_FAILED	Locking failed.
168 ERROR_SWAPIO_FAILED	Swap IO failed.
169 ERROR_SWAPIN_FAILED	Swap in failed.
170 ERROR_BUSY	Busy.
180 ERROR_INVALID_SEGMENT_NUMBER	Invalid segment number.
181 ERROR_INVALID_CALLGATE	Invalid call gate.
182 ERROR_INVALID_ORDINAL	Invalid ordinal.
183 ERROR_ALREADY_EXISTS	Shared segment already exists.
184 ERROR_NO_CHILD_PROCESS	No child process to wait for.
185 ERROR_CHILD_ALIVE_NOWAIT	NoWait specified and child alive.
186 ERROR_INVALID_FLAG_NUMBER	Invalid flag number.
187 ERROR_SEM_NOT_FOUND	Semaphore does not exist.
188 ERROR_INVALID_STARTING_CODESEG	Invalid starting code segment, incorrect END (label) directive.
189 ERROR_INVALID_STACKSEG	Invalid stack segment.
190 ERROR_INVALID_MODULETYPE	Invalid module type — dynamic-link library file cannot be used as an application. Application cannot be used as a dynamic-link library.
191 ERROR_INVALID_EXE_SIGNATURE	Invalid EXE signature — file is DOS mode program or improper program.
192 ERROR_EXE_MARKED_INVALID	EXE marked invalid — link detected errors when application created.
193 ERROR_BAD_EXE_FORMAT	Bad EXE format — file is DOS mode program or improper program.
194 ERROR_ITERATED_DATA_EXCEEDS_64K	Iterated data exceeds 64KB — more than 64KB of data in one of the segments of the file.
195 ERROR_INVALID_MINALLOCSIZE	Invalid minimum allocation size — size is specified to be less than the size of the segment data in the file.
196 ERROR_DYNLINK_FROM_INVALID_RING	Dynamic link from invalid privilege level — privilege level 2 routine cannot link to dynamic-link libraries.
197 ERROR_IOPL_NOT_ENABLED	IOPL not enabled — IOPL set to “NO” in CONFIG.SYS.
198 ERROR_INVALID_SEGDPL	Invalid segment descriptor privilege level — can only have privilege levels of 2 and 3.
199 ERROR_AUTODATASEG_EXCEEDS_64k	Automatic data segment exceeds 64KB.
200 ERROR_RING2SEG_MUST_BE_MOVABLE	Privilege level 2 segment must be movable.
201 ERROR_RELOC_CHAIN_XEEDS_SEGLIM	Relocation chain exceeds segment limit.
202 ERROR_INFLOOP_IN_RELOC_CHAIN	Infinite loop in relocation chain segment.
203 ERROR_ENVVAR_NOT_FOUND	Environment variable not found.
204 ERROR_NOT_CURRENT_CTRY	Not current country.
205 ERROR_NO_SIGNAL_SENT	No signal sent — no process in the command subtree has a signal handler.
206 ERROR_FILENAME_EXCED_RANGE	File name or extension greater than “8.3” characters.
207 ERROR_RING2_STACK_IN_USE	Privilege level 2 stack in use.
208 ERROR_META_EXPANSION_TOO_LONG	Meta (global) expansion is too long.
209 ERROR_INVALID_SIGNAL_NUMBER	Invalid signal number.
210 ERROR_THREAD_1_INACTIVE	Inactive thread.
211 ERROR_INFO_NOT_AVAIL	File system information not available for this file.
212 ERROR_LOCKED	Locked error.
213 ERROR_BAD_DYNALINK	Attempted to execute non-family API in DOS mode.
214 ERROR_TOO_MANY_MODULES	Too many modules.
215 ERROR_NESTING_NOT_ALLOWED	Nesting not allowed.
217 ERROR_ZOMBIE_PROCESS	Zombie process.
218 ERROR_STACK_IN_HIGH_MEMORY	Stack in high memory.
219 ERROR_INVALID_EXITROUTINE_RING	Invalid exit routine ring.
220 ERROR_GETBUF_FAILED	Get buffer failed.

221 ERROR_FLUSHBUF_FAILED	Flush buffer failed.
222 ERROR_TRANSFER_TOO_LONG	Transfer is too long.
228 ERROR_NO_CHILDREN	No child process.
229 ERROR_INVALID_SCREEN_GROUP	Invalid session.
230 ERROR_BAD_PIPE	Non-existent pipe or bad operation.
231 ERROR_PIPE_BUSY	Pipe is busy.
232 ERROR_NO_DATA	No data available on non-blocking read.
233 ERROR_PIPE_NOT_CONNECTED	Pipe was disconnected by server.
234 ERROR_MORE_DATA	More data is available.
240 ERROR_VC_DISCONNECTED	Session was dropped due to errors.
250 ERROR_CIRCULARITY_REQUESTED	Renaming a directory that would cause a circularity problem.
251 ERROR_DIRECTORY_IN_CDS	Renaming a directory that is in use.
252 ERROR_INVALID_FSD_NAME	Trying to access nonexistent FSD.
253 ERROR_INVALID_PATH	Bad pseudo device.
254 ERROR_INVALID_EA_NAME	Bad character in name, or bad cbName.
255 ERROR_EA_LIST_INCONSISTENT	List does not match its size, or bad EAs in list.
256 ERROR_EA_LIST_TOO_LONG	FEAList > 64K-1 bytes.
257 ERROR_NO_META_MATCH	String doesn't match expression.
259 ERROR_NO_MORE_ITEMS	DosQFSAttach ordinal query.
260 ERROR_SEARCH_STRUC_REUSED	DOS mode findfirst/next search structure reused.
261 ERROR_CHAR_NOT_FOUND	Character not found.
262 ERROR_TOO_MUCH_STACK	Stack request exceeds system limit.
263 ERROR_INVALID_ATTR	Invalid attribute.
264 ERROR_INVALID_STARTING_RING	Invalid starting ring.
265 ERROR_INVALID_DLL_INIT_RING	Invalid DLL INIT ring.
266 ERROR_CANNOT_COPY	Cannot copy.
267 ERROR_DIRECTORY	Used by DOSCOPY in doscall1.
268 ERROR_OPLOCKED_FILE	Oplocked file.
269 ERROR_OPLOCK_THREAD_EXISTS	Oplock thread exists.
270 ERROR_VOLUME_CHANGED	Volume changed.
271 - 273	Reserved.
274 ERROR_ALREADY_SHUTDOWN	System already shutdown.
275 ERROR_EAS_DIDNT_FIT	EAS didnt fit.
303 ERROR_INVALID_PROCID	Invalid process identity.
304 ERROR_INVALID_PDELTA	Invalid priority delta.
305 ERROR_NOT_DESCENDANT	Not descendant.
306 ERROR_NOT_SESSION_MANAGER	Requestor not session manager.
307 ERROR_INVALID_PCLASS	Invalid P class.
308 ERROR_INVALID_SCOPE	Invalid scope.
309 ERROR_INVALID_THREADID	Invalid thread identity.
310 ERROR_DOSSUB_SHRINK	Cannot shrink segment - DosSubSet.
311 ERROR_DOSSUB_NOMEM	No memory to satisfy request - DosSubAlloc.
312 ERROR_DOSSUB_OVERLAP	Overlap of specified block with an allocated memory - DosSubFree.
313 ERROR_DOSSUB_BADSIZE	Bad size parameter - DosSubAlloc or DosSubFree.
314 ERROR_DOSSUB_BADFLAG	Bad flag parameter - DosSubSet.
315 ERROR_DOSSUB_BADSELECTOR	Invalid segment selector.
316 ERROR_MR_MSG_TOO_LONG	Message too long for buffer.
317 ERROR_MR_MID_NOT_FOUND	Message identity number not found.
318 ERROR_MR_UN_ACC_MSGF	Unable to access message file.
319 ERROR_MR_INV_MSGF_FORMAT	Invalid message file format.
320 ERROR_MR_INV_IVCOUNT	Invalid insertion variable count.
321 ERROR_MR_UN_PERFORM	Unable to perform function.
322 ERROR_TS_WAKEUP	Unable to wake up.
323 ERROR_TS_SEMHANDLE	Invalid system semaphore.
324 ERROR_TS_NOTIMER	No timers available.
326 ERROR_TS_HANDLE	Invalid timer handle.
327 ERROR_TS_DATETIME	Date or time invalid.
328 ERROR_SYS_INTERNAL	Internal system error.
329 ERROR_QUE_CURRENT_NAME	Current queue name does not exist.
330 ERROR_QUE_PROC_NOT_OWNED	Current process does not own queue.
331 ERROR_QUE_PROC_OWNED	Current process owns queue.

332 ERROR_QUE_DUPLICATE	Duplicate queue name.
333 ERROR_QUE_ELEMENT_NOT_EXIST	Queue element does not exist.
334 ERROR_QUE_NO_MEMORY	Inadequate queue memory.
335 ERROR_QUE_INVALID_NAME	Invalid queue name.
336 ERROR_QUE_INVALID_PRIORITY	Invalid queue priority parameter.
337 ERROR_QUE_INVALID_HANDLE	Invalid queue handle.
338 ERROR_QUE_LINK_NOT_FOUND	Queue link not found.
339 ERROR_QUE_MEMORY_ERROR	Queue memory error.
340 ERROR_QUE_PREV_AT_END	Previous queue element was at end of queue.
341 ERROR_QUE_PROC_NO_ACCESS	Process does not have access to queues.
342 ERROR_QUE_EMPTY	Queue is empty.
343 ERROR_QUE_NAME_NOT_EXIST	Queue name does not exist.
344 ERROR_QUE_NOT_INITIALIZED	Queues not initialized.
345 ERROR_QUE_UNABLE_TO_ACCESS	Unable to access queues.
346 ERROR_QUE_UNABLE_TO_ADD	Unable to add new queue.
347 ERROR_QUE_UNABLE_TO_INIT	Unable to initialize queues.
349 ERROR_VIO_INVALID_MASK	Invalid function replaced.
350 ERROR_VIO_PTR	Invalid pointer to parameter.
351 ERROR_VIO_APTR	Invalid pointer to attribute.
352 ERROR_VIO_RPTR	Invalid pointer to row.
353 ERROR_VIO_CPTR	Invalid pointer to column.
354 ERROR_VIO_LPTR	Invalid pointer to length.
355 ERROR_VIO_MODE	Unsupported screen mode.
356 ERROR_VIO_WIDTH	Invalid cursor width value.
357 ERROR_VIO_ATTR	Invalid cursor attribute value.
358 ERROR_VIO_ROW	Invalid row value.
359 ERROR_VIO_COL	Invalid column value.
360 ERROR_VIO_TOPROW	Invalid TopRow value.
361 ERROR_VIO_BOTROW	Invalid BotRow value.
362 ERROR_VIO_RIGHTCOL	Invalid right column value.
363 ERROR_VIO_LEFTCOL	Invalid left column value.
364 ERROR_SCS_CALL	Call issued by other than sm
365 ERROR_SCS_VALUE	Value is not for save or restore.
366 ERROR_VIO_WAIT_FLAG	Invalid wait flag setting.
367 ERROR_VIO_UNLOCK	Screen not previously locked.
368 ERROR_SGS_NOT_SESSION_MGR	Caller not session manager.
369 ERROR_SMG_INVALID_SGID	Invalid session identity.
369 ERROR_SMG_INVALID_SESSION_ID	Invalid session ID.
370 ERROR_SMG_NOSG	No sessions available.
370 ERROR_SMG_NO_SESSIONS	No sessions available.
371 ERROR_SMG_GRP_NOT_FOUND	Session not found.
371 ERROR_SMG_SESSION_NOT_FOUND	Session not found.
372 ERROR_SMG_SET_TITLE	Title sent by shell or parent cannot be changed.
373 ERROR_KBD_PARAMETER	Invalid parameter to keyboard.
374 ERROR_KBD_NO_DEVICE	No device.
375 ERROR_KBD_INVALID_IOWAIT	Invalid I/O wait specified.
376 ERROR_KBD_INVALID_LENGTH	Invalid length for keyboard.
377 ERROR_KBD_INVALID_ECHO_MASK	Invalid echo mode mask.
378 ERROR_KBD_INVALID_INPUT_MASK	Invalid input mode mask.
379 ERROR_MON_INVALID_PARMS	Invalid parameters to DosMon.
380 ERROR_MON_INVALID_DEVNAME	Invalid device name string.
381 ERROR_MON_INVALID_HANDLE	Invalid device handle.
382 ERROR_MON_BUFFER_TOO_SMALL	Buffer too small.
383 ERROR_MON_BUFFER_EMPTY	Buffer is empty.
384 ERROR_MON_DATA_TOO_LARGE	Data record too large.
385 ERROR_MOUSE_NO_DEVICE	Mouse device closed (invalid device handle).
386 ERROR_MOUSE_INV_HANDLE	Mouse device closed (invalid device handle).
387 ERROR_MOUSE_INV_PARMS	Parameters invalid for display mode.
388 ERROR_MOUSE_CANT_RESET	Function assigned and cannot be reset.
389 ERROR_MOUSE_DISPLAY_PARMS	Parameters invalid for display mode.
390 ERROR_MOUSE_INV_MODULE	Module not valid.
391 ERROR_MOUSE_INV_ENTRY_PT	Entry point not valid.
392 ERROR_MOUSE_INV_MASK	Function mask invalid.

393 NO_ERROR_MOUSE_NO_DATA	No valid data.
394 NO_ERROR_MOUSE_PTR_DRAWN	Pointer drawn.
395 ERROR_INVALID_FREQUENCY	Invalid frequency for beep.
396 ERROR-NLS_NO_COUNTRY_FILE	Cannot find COUNTRY.SYS file.
397 ERROR-NLS_OPEN_FAILED	Cannot open COUNTRY.SYS file.
398 ERROR-NLS_NO_CTRY_CODE	Country code not found.
398 ERROR_NO_COUNTRY_OR_CODEPAGE	Country code not found.
399 ERROR-NLS_TABLE_TRUNCATED	Table returned information truncated, buffer too small.
400 ERROR-NLS_BAD_TYPE	Selected type does not exist.
401 ERROR-NLS_TYPE_NOT_FOUND	Selected type not in file.
402 ERROR_VIO_SMG_ONLY	Valid from session manager only.
403 ERROR_VIO_INVALID_ASCIIIZ	Invalid ASCIIIZ length.
404 ERROR_VIO_DEREGISTER	VioDeRegister not allowed.
405 ERROR_VIO_NO_POPUP	Pop-up window not allocated.
406 ERROR_VIO_EXISTING_POPUP	Pop-up window on screen (NoWait).
407 ERROR_KBD_SMG_ONLY	Valid from session manager only.
408 ERROR_KBD_INVALID_ASCIIIZ	Invalid ASCIIIZ length.
409 ERROR_KBD_INVALID_MASK	Invalid replacement mask.
410 ERROR_KBD_REGISTER	KbdRegister not allowed.
411 ERROR_KBD_DEREGISTER	KbdDeRegister not allowed.
412 ERROR_MOUSE_SMG_ONLY	Valid from session manager only.
413 ERROR_MOUSE_INVALID_ASCIIIZ	Invalid ASCIIIZ length.
414 ERROR_MOUSE_INVALID_MASK	Invalid replacement mask.
415 ERROR_MOUSE_REGISTER	Mouse register not allowed.
416 ERROR_MOUSE_DEREGISTER	Mouse deregister not allowed.
417 ERROR_SMG_BAD_ACTION	Invalid action specified.
418 ERROR_SMG_INVALID_CALL	INIT called more than once or invalid session identity.
419 ERROR_SCS_SG_NOTFOUND	New session number.
420 ERROR_SCS_NOT_SHELL	Caller is not shell.
421 ERROR_VIO_INVALID_PARMS	Invalid parameters passed.
422 ERROR_VIO_FUNCTION_OWNED	Save/restore already owned.
423 ERROR_VIO_RETURN	Non-destruct return (undo).
424 ERROR_SCS_INVALID_FUNCTION	Caller invalid function.
425 ERROR_SCS_NOT_SESSION_MGR	Caller not session manager.
426 ERROR_VIO_REGISTER	Vio register not allowed.
427 ERROR_VIO_NO_MODE_THREAD	No mode restore thread in SG.
428 ERROR_VIO_NO_SAVE_RESTORE_THD	No save/rest thread in SG.
429 ERROR_VIO_IN_BG	Function invalid in background.
430 ERROR_VIO_ILLEGAL_DURING_POPUP	Function not allowed during pop-up window.
431 ERROR_SMG_NOT_BASESHELL	Caller is not the base shell.
432 ERROR_SMG_BAD_STATUSREQ	Invalid status requested.
433 ERROR_QUE_INVALID_WAIT	NoWait parameter out of bounds.
434 ERROR_VIO_LOCK	Error returned from Scroll Lock.
435 ERROR_MOUSE_INVALID_IOWAIT	Invalid parameters for IOWait.
436 ERROR_VIO_INVALID_HANDLE	Invalid VIO handle.
437 ERROR_VIO_ILLEGAL_DURING_LOCK	Function not allowed during screen lock.
438 ERROR_VIO_INVALID_LENGTH	Invalid VIO length.
439 ERROR_KBD_INVALID_HANDLE	Invalid KBD handle.
440 ERROR_KBD_NO_MORE_HANDLE	Ran out of handles.
441 ERROR_KBD_CANNOT_CREATE_KCB	Unable to create kcb.
442 ERROR_KBD_CODEPAGE_LOAD_INCOMPL	Unsuccessful code-page load.
443 ERROR_KBD_INVALID_CODEPAGE_ID	Invalid code-page identity.
444 ERROR_KBD_NO_CODEPAGE_SUPPORT	No code page support.
445 ERROR_KBD_FOCUS_REQUIRED	Keyboard focus required.
446 ERROR_KBD_FOCUS_ALREADY_ACTIVE	Calling thread has an outstanding focus.
447 ERROR_KBD_KEYBOARD_BUSY	Keyboard busy.
448 ERROR_KBD_INVALID_CODEPAGE	Invalid code page.
449 ERROR_KBD_UNABLE_TO_FOCUS	Focus attempt failed.
450 ERROR_SMG_SESSION_NON_SELECT	Session is not selectable.
451 ERROR_SMG_SESSION_NOT_FOREGRND	Parent/child session not foreground.
452 ERROR_SMG_SESSION_NOT_PARENT	Not parent of requested child.
453 ERROR_SMG_INVALID_START_MODE	Invalid session start mode.
454 ERROR_SMG_INVALID_RELATED_OPT	Invalid session start related option.

455	ERROR_SMG_INVALID_BOND_OPTION	Invalid session bond option.
456	ERROR_SMG_INVALID_SELECT_OPT	Invalid session select option.
457	ERROR_SMG_START_IN_BACKGROUND	Session started in background.
458	ERROR_SMG_INVALID_STOP_OPTION	Invalid session stop option.
459	ERROR_SMG_BAD_RESERVE	Reserved parameters not zero.
460	ERROR_SMG_PROCESS_NOT_PARENT	Session parent process already exists.
461	ERROR_SMG_INVALID_DATA_LENGTH	Invalid data length.
462	ERROR_SMG_NOT_BOUND	Parent not bound.
463	ERROR_SMG_RETRY_SUB_ALLOC	Retry request block allocation.
464	ERROR_KBD_DETACHED	This call not allowed for detached PID.
465	ERROR_VIO_DETACHED	This call disallowed for detached pid.
466	ERROR_MOU_DETACHED	This call disallowed for detached pid.
467	ERROR_VIO_FONT	No font available to support mode.
468	ERROR_VIO_USER_FONT	User font active.
469	ERROR_VIO_BAD_CP	Invalid code page specified.
470	ERROR_VIO_NO_CP	System displays do not support code page.
471	ERROR_VIO_NA_CP	Current display does not support code page.
472	ERROR_INVALID_CODE_PAGE	Invalid code page.
473	ERROR_CPLIST_TOO_SMALL	Code page list is too small.
474	ERROR_CP_NOT_MOVED	Code page not moved.
475	ERROR_MODE_SWITCH_INIT	Mode switch initialization error.
476	ERROR_CODE_PAGE_NOT_FOUND	Code page not found.
477	ERROR_UNEXPECTED_SLOT_RETURNED	Internal error.
478	ERROR_SMG_INVALID_TRACE_OPTION	Invalid start session trace indicator.
479	ERROR_VIO_INTERNAL_RESOURCE	VIO internal resource error.
480	ERROR_VIO_SHELL_INIT	VIO shell initialization error.
481	ERROR_SMG_NO_HARD_ERRORS	No session manager hard errors.
482	ERROR_CP_SWITCH_INCOMPLETE	DosSetCp unable to set KBD or VIO code page.
483	ERROR_VIO_TRANSPARENT_POPUP	Error during VIO pop-up window.
484	ERROR_CRITSEC_OVERFLOW	Critical section overflow.
485	ERROR_CRITSEC_UNDERFLOW	Critical section underflow.
486	ERROR_VIO_BAD_RESERVE	Reserved parameter is not zero.
487	ERROR_INVALID_ADDRESS	Bad physical address.
488	ERROR_ZERO_SELECTORS_REQUESTED	At least one selector must be requested.
489	ERROR_NOT_ENOUGH_SELECTORS_AVA	Not enough GDT selectors to satisfy request.
490	ERROR_INVALID_SELECTOR	Not a GDT selector.
491	ERROR_SMG_INVALID_PROGRAM_TYPE	Invalid program type.
492	ERROR_SMG_INVALID_PGM_CONTROL	Invalid program control.
493	ERROR_SMG_INVALID_INHERIT_OPT	Bad inherit option.
494	ERROR_VIO_EXTENDED_SG	
495	ERROR_VIO_NOT_PRES_MGR_SG	
496	ERROR_VIO_SHIELD_OWNED	
497	ERROR_VIO_NO_MORE_HANDLES	
498	ERROR_VIO_SEE_ERROR_LOG	
499	ERROR_VIO_ASSOCIATED_DC	
500	ERROR_KBD_NO_CONSOLE	
501	ERROR_MOUSE_NO_CONSOLE	
502	ERROR_MOUSE_INVALID_HANDLE	
503	ERROR_SMG_INVALID_DEBUG_PARMS	
504	ERROR_KBD_EXTENDED_SG	
505	ERROR_MOU_EXTENDED_SG	
506	ERROR_SMG_INVALID_ICON_FILE	

## Alphabetic Order

Error Name and Number	Description
ERROR_ACCESS_DENIED 5	Access denied.
ERROR_ALREADY_ASSIGNED 85	Duplicate redirection.
ERROR_ALREADY_EXISTS 183	Shared segment already exists.
ERROR_ALREADY_SHUTDOWN 274	System already shutdown.
ERROR_ARENA_TRASHED 7	Memory control blocks destroyed.
ERROR_AUTODATASEG_EXCEEDS_64k 199	Automatic data segment exceeds 64KB.
ERROR_BAD_ARGUMENTS 160	Bad environment pointer.

ERROR_BAD_COMMAND	22	Unknown command.
ERROR_BAD_DRIVER_LEVEL	119	Level four driver not found.
ERROR_BAD_DYNALINK	213	Attempted to execute non-family API in DOS mode.
ERROR_BAD_ENVIRONMENT	10	Invalid environment.
ERROR_BAD_EXE_FORMAT	193	Bad EXE format — file is DOS mode program or improper program.
ERROR_BAD_FORMAT	11	Invalid format.
ERROR_BAD_LENGTH	24	Bad request structure length.
ERROR_BAD_PATHNAME	161	Bad path name passed to exec.
ERROR_BAD_PIPE	230	Non-existent pipe or bad operation.
ERROR_BAD_THREADID_ADDR	159	Bad thread-identity address.
ERROR_BAD_UNIT	20	Unknown unit.
ERROR_BROKEN_PIPE	109	Write on pipe with no reader.
ERROR_BUFFER_OVERFLOW	111	Buffer passed to system call too small to hold return data.
ERROR_BUSY	170	Busy.
ERROR_BUSY_DRIVE	142	Specified drive is busy.
ERROR_CALL_NOT_IMPLEMENTED	120	Invalid function called.
ERROR_CANNOT_COPY	266	Cannot copy.
ERROR_CANNOT_MAKE	82	Cannot make directory entry.
ERROR_CHAR_NOT_FOUND	261	Character not found.
ERROR_CHILD_ALIVE_NOWAIT	185	NoWait specified and child alive.
ERROR_CHILD_NOT_COMPLETE	129	DosCwait children not terminated.
ERROR_CIRCULARITY_REQUESTED	250	Renaming a directory that would cause a circularity problem.
ERROR_CODE_PAGE_NOT_FOUND	476	Code page not found.
ERROR_CPLIST_TOO_SMALL	473	Code page list is too small.
ERROR_CP_NOT_MOVED	474	Code page not moved.
ERROR_CP_SWITCH_INCOMPLETE	482	DosSetCp unable to set KBD or VIO code page.
ERROR_CRC	23	Data error (CRC).
ERROR_CRITSEC_OVERFLOW	484	Critical section overflow.
ERROR_CRITSEC_UNDERFLOW	485	Critical section underflow.
ERROR_CURRENT_DIRECTORY	16	Attempting to remove current directory.
ERROR_DEVICE_IN_USE	99	Device in use.
ERROR_DIRECTORY	267	Used by DOSCOPY in doscall1.
ERROR_DIRECTORY_IN_CDS	251	Renaming a directory that is in use.
ERROR_DIRECT_ACCESS_HANDLE	130	Handle operation invalid for direct disk-access handles.
ERROR_DIR_NOT_EMPTY	145	Directory must be empty to use join command.
ERROR_DIR_NOT_ROOT	144	Directory must be a subdirectory of the root.
ERROR_DISCARDED	157	Segment is discarded.
ERROR_DISK_CHANGE	107	Insert drive B disk into drive A.
ERROR_DISK_FULL	112	Not enough space on the disk.
ERROR_DOSSUB_BADFLAG	314	Bad flag parameter — DosSubSet.
ERROR_DOSSUB_BADSELECTOR	315	Invalid segment selector.
ERROR_DOSSUB_BADSIZE	313	Bad size parameter — DosSubAlloc or DosSubFree.
ERROR_DOSSUB_NOMEM	311	No memory to satisfy request — DosSubAlloc.
ERROR_DOSSUB_OVERLAP	312	Overlap of specified block with an allocated memory — DosSubFree.
ERROR_DOSSUB_SHRINK	310	Cannot shrink segment — DosSubSet.
ERROR_DRIVE_LOCKED	108	Drive locked by another process.
ERROR_DUP_FCB	81	Reserved.
ERROR_DYNLINK_FROM_INVALID_RING	196	Dynamic link from invalid privilege level — privilege level 2 routine cannot link to dynamic-link libraries.
ERROR_EA_LIST_INCONSISTENT	255	List does not match its size, or bad EAs in list.
ERROR_EA_LIST_TOO_LONG	256	FEAList > 64K—1 bytes.
ERROR_EAS_DIDNT_FIT	275	EAS didnt fit.
ERROR_ENVVAR_NOT_FOUND	203	Environment variable not found.
ERROR_EXCL_SEM_ALREADY_OWNED	101	Exclusive semaphore already owned.
ERROR_EXE_MARKED_INVALID	192	EXE marked invalid — link detected errors when application created.
ERROR_FAIL_I24	83	Fail on INT 24.
ERROR_FCB_UNAVAILABLE	35	FCB unavailable.
ERROR_FILENAME_EXCED_RANGE	206	File name or extension greater than "8.3" characters.
ERROR_FILE_EXISTS	80	File exists.

ERROR_FILE_NOT_FOUND	2	File not found.
ERROR_FLUSHBUF_FAILED	221	Flush buffer failed.
ERROR_GEN_FAILURE	31	General failure.
ERROR_GETBUF_FAILED	220	Get buffer failed.
ERROR_INFLOOP_IN_RELOC_CHAIN	202	Infinite loop in relocation chain segment.
ERROR_INFO_NOT_AVAIL	211	File system information not available for this file.
ERROR_INSUFFICIENT_BUFFER	122	Data buffer too small.
ERROR_INTERRUPT	95	Interrupted system call.
ERROR_INVALID_ACCESS	12	Invalid access code.
ERROR_INVALID_ADDRESS	487	Bad physical address.
ERROR_INVALID_ATTR	263	Invalid attribute.
ERROR_INVALID_AT_INTERRUPT_TIME	104	Operation invalid at interrupt time.
ERROR_INVALID_BLOCK	9	Invalid memory-block address.
ERROR_INVALID_CALLGATE	181	Invalid call gate.
ERROR_INVALID_CATEGORY	117	Category for DevIOCtl not defined.
ERROR_INVALID_CODE_PAGE	472	Invalid code page.
ERROR_INVALID_DATA	13	Invalid data.
ERROR_INVALID_DLL_INIT_RING	265	Invalid DLL INIT ring.
ERROR_INVALID_DRIVE	15	Invalid drive specified.
ERROR_INVALID_EA_NAME	254	Bad character in name, or bad cbName.
ERROR_INVALID_EVENT_COUNT	151	DosMuxSemWait errors.
ERROR_INVALID_EXE_SIGNATURE	191	Invalid EXE signature — file is DOS mode program or improper program.
ERROR_INVALID_EXITROUTINE_RING	219	Invalid exit routine ring.
ERROR_INVALID_FLAG_NUMBER	186	Invalid flag number.
ERROR_INVALID_FREQUENCY	395	Invalid frequency for beep.
ERROR_INVALID_FSD_NAME	252	Trying to access nonexistent FSD.
ERROR_INVALID_FUNCTION	1	Invalid function number.
ERROR_INVALID_HANDLE	6	Invalid handle.
ERROR_INVALID_LEVEL	124	Non-implemented level for information retrieval or setting.
ERROR_INVALID_LIST_FORMAT	153	Invalid list format.
ERROR_INVALID_MINALLOCSIZE	195	Invalid minimum allocation size — size is specified to be less than the size of the segment data in the file.
ERROR_INVALID_MODULETYPE	190	Invalid module type — dynamic-link library file cannot be used as an application. Application cannot be used as a dynamic-link library.
ERROR_INVALID_NAME	123	Illegal character or bad file-system name.
ERROR_INVALID_ORDINAL	182	Invalid ordinal.
ERROR_INVALID_PARAMETER	87	Invalid parameter.
ERROR_INVALID_PASSWORD	86	Invalid password.
ERROR_INVALID_PATH	253	Bad pseudo device.
ERROR_INVALID_PCLASS	307	Invalid P class.
ERROR_INVALID_PDELTA	304	Invalid priority delta.
ERROR_INVALID_PROCID	303	Invalid process identity.
ERROR_INVALID_SCOPE	308	Invalid scope.
ERROR_INVALID_SCREEN_GROUP	229	Invalid session.
ERROR_INVALID_SEGDPL	198	Invalid segment descriptor privilege level — can only have privilege levels of 2 and 3.
ERROR_INVALID_SEGMENT_NUMBER	180	Invalid segment number.
ERROR_INVALID_SELECTOR	490	Not a GDT selector.
ERROR_INVALID_SIGNAL_NUMBER	209	Invalid signal number.
ERROR_INVALID_STACKSEG	189	Invalid stack segment.
ERROR_INVALID_STARTING_CODESEG	188	Invalid starting code segment, incorrect END (label) directive.
ERROR_INVALID_STARTING_RING	264	Invalid starting ring.
ERROR_INVALID_TARGET_HANDLE	114	Target handle in DosDupHandle invalid.
ERROR_INVALID_THREADID	309	Invalid thread identity.
ERROR_INVALID_VERIFY_SWITCH	118	Invalid value passed for verify flag.
ERROR_IOPL_NOT_ENABLED	197	IOPL not enabled — IOPL set to "NO" in CONFIG.SYS.
ERROR_IS_JOINED	134	Drive is already joined.
ERROR_IS_JOIN_PATH	147	Path specified is being used in join.
ERROR_IS_JOIN_TARGET	133	Drive has previously joined drives.
ERROR_IS_SUBSTED	135	Drive is already substituted.



ERROR_IS_SUBST_PATH	146	Path specified is being used in a substitute.
ERROR_IS_SUBST_TARGET	149	Cannot join or substitute drive having directory that is target of a previous substitute.
ERROR_ITERATED_DATA_EXCEEDS_64K	194	Iterated data exceeds 64KB — more than 64KB of data in one of the segments of the file.
ERROR_JOIN_TO_JOIN	138	Cannot join to a joined drive.
ERROR_JOIN_TO_SUBST	140	Cannot join to a substituted drive.
ERROR_KBD_CANNOT_CREATE_KCB	441	Unable to create kcb.
ERROR_KBD_CODEPAGE_LOAD_INCOMPL	442	Unsuccessful code-page load.
ERROR_KBD_DEREGISTER	411	KbdDeRegister not allowed.
ERROR_KBD_DETACHED	464	This call not allowed for detached PID.
ERROR_KBD_EXTENDED_SG	504	
ERROR_KBD_FOCUS_ALREADY_ACTIVE	446	Calling thread has an outstanding focus.
ERROR_KBD_FOCUS_REQUIRED	445	Keyboard focus required.
ERROR_KBD_INVALID_ASCIIIZ	408	Invalid ASCIIIZ length.
ERROR_KBD_INVALID_CODEPAGE	448	Invalid code page.
ERROR_KBD_INVALID_CODEPAGE_ID	443	Invalid code-page identity.
ERROR_KBD_INVALID_ECHO_MASK	377	Invalid echo mode mask.
ERROR_KBD_INVALID_HANDLE	439	Invalid KBD handle.
ERROR_KBD_INVALID_INPUT_MASK	378	Invalid input mode mask.
ERROR_KBD_INVALID_IOWAIT	375	Invalid I/O wait specified.
ERROR_KBD_INVALID_LENGTH	376	Invalid length for keyboard.
ERROR_KBD_INVALID_MASK	409	Invalid replacement mask.
ERROR_KBD_KEYBOARD_BUSY	447	Keyboard busy.
ERROR_KBD_NO_CODEPAGE_SUPPORT	444	No code page support.
ERROR_KBD_NO_CONSOLE	500	
ERROR_KBD_NO_DEVICE	374	No device.
ERROR_KBD_NO_MORE_HANDLE	440	Ran out of handles.
ERROR_KBD_PARAMETER	373	Invalid parameter to keyboard.
ERROR_KBD_REGISTER	410	KbdRegister not allowed.
ERROR_KBD_SMG_ONLY	407	Valid from session manager only.
ERROR_KBD_UNABLE_TO_FOCUS	449	Focus attempt failed.
ERROR_LABEL_TOO_LONG	154	Volume label too big.
ERROR_LOCKED	212	Locked error.
ERROR_LOCK_FAILED	167	Locking failed.
ERROR_LOCK_VIOLATION	33	Lock violation.
ERROR_MAX_THRDS_REACHED	164	No more process slots.
ERROR_META_EXPANSION_TOO_LONG	208	Meta (global) expansion is too long.
ERROR_MODE_SWITCH_INIT	475	Mode switch initialization error.
ERROR_MOD_NOT_FOUND	126	Module handle not found with getprocaddr, getmodhandle.
ERROR_MONITORS_NOT_SUPPORTED	165	ERROR_I24 mapping.
ERROR_MON_BUFFER_EMPTY	383	Buffer is empty.
ERROR_MON_BUFFER_TOO_SMALL	382	Buffer too small.
ERROR_MON_DATA_TOO_LARGE	384	Data record too large.
ERROR_MON_INVALID_DEVNAME	380	Invalid device name string.
ERROR_MON_INVALID_HANDLE	381	Invalid device handle.
ERROR_MON_INVALID_PARMS	379	Invalid parameters to DosMon.
ERROR_MORE_DATA	234	More data is available.
ERROR_MOUSE_CANT_RESET	388	Function assigned and cannot be reset.
ERROR_MOUSE_DEREGISTER	416	Mouse deregister not allowed.
ERROR_MOUSE_DISPLAY_PARMS	389	Parameters invalid for display mode.
ERROR_MOUSE_INVALID_ASCIIIZ	413	Invalid ASCIIIZ length.
ERROR_MOUSE_INVALID_HANDLE	502	
ERROR_MOUSE_INVALID_IOWAIT	435	Invalid parameters for IOWait.
ERROR_MOUSE_INVALID_MASK	414	Invalid replacement mask.
ERROR_MOUSE_INV_ENTRY_PT	391	Entry point not valid.
ERROR_MOUSE_INV_HANDLE	386	Mouse device closed (invalid device handle).
ERROR_MOUSE_INV_MASK	392	Function mask invalid.
ERROR_MOUSE_INV_MODULE	390	Module not valid.
ERROR_MOUSE_INV_PARMS	387	Parameters invalid for display mode.
ERROR_MOUSE_NO_CONSOLE	501	
ERROR_MOUSE_NO_DEVICE	385	Mouse device closed (invalid device handle).
ERROR_MOUSE_REGISTER	415	Mouse register not allowed.

ERROR_MOUSE_SMG_ONLY	412	Valid from session manager only.
ERROR_MOU_DETACHED	466	This call disallowed for detached pid.
ERROR_MOU_EXTENDED_SG	505	
ERROR_MR_INV_IVCOUNT	320	Invalid insertion variable count.
ERROR_MR_INV_MSGF_FORMAT	319	Invalid message file format.
ERROR_MR_MID_NOT_FOUND	317	Message identity number not found.
ERROR_MR_MSG_TOO_LONG	316	Message too long for buffer.
ERROR_MR_UN_ACC_MSGF	318	Unable to access message file.
ERROR_MR_UN_PERFORM	321	Unable to perform function.
ERROR_NEGATIVE_SEEK	131	Attempting seek to negative offset.
ERROR_NESTING_NOT_ALLOWED	215	Nesting not allowed.
ERROR_NET_WRITE_FAULT	88	Network device fault.
ERROR_NLS_BAD_TYPE	400	Selected type does not exist.
ERROR_NLS_NO_COUNTRY_FILE	396	Cannot find COUNTRY.SYS file.
ERROR_NLS_NO_CTRY_CODE	398	Country code not found.
ERROR_NLS_OPEN_FAILED	397	Cannot open COUNTRY.SYS file.
ERROR_NLS_TABLE_TRUNCATED	399	Table returned information truncated, buffer too small.
ERROR_NLS_TYPE_NOT_FOUND	401	Selected type not in file.
ERROR_NOT_CURRENT_CTRY	204	Not current country.
ERROR_NOT_DESCENDANT	305	Not descendant.
ERROR_NOT_DOS_DISK	26	Unknown media type.
ERROR_NOT_ENOUGH_MEMORY	8	Insufficient memory.
ERROR_NOT_ENOUGH_SELECTORS_AVA	489	Not enough GDT selectors to satisfy request.
ERROR_NOT_FROZEN	90	System error.
ERROR_NOT_JOINED	136	Cannot delete drive that is not joined.
ERROR_NOT_LOCKED	158	Segment not locked.
ERROR_NOT_READY	21	Drive not ready.
ERROR_NOT_SAME_DEVICE	17	Not same device.
ERROR_NOT_SESSION_MANAGER	306	Requestor not session manager.
ERROR_NOT_SUBSTED	137	Cannot delete drive that is not substituted.
ERROR_NOT_SUPPORTED	50	Network request not supported.
ERROR_NO_CHILDREN	228	No child process.
ERROR_NO_CHILD_PROCESS	184	No child process to wait for.
ERROR_NO_COUNTRY_OR_CODEPAGE	398	Country code not found.
ERROR_NO_DATA	232	No data available on non-blocking read.
ERROR_NO_ITEMS	93	No items to work on.
ERROR_NO_META_MATCH	257	String doesn't match expression.
ERROR_NO_MORE_FILES	18	No more files.
ERROR_NO_MORE_ITEMS	259	DosQFSAttach ordinal query.
ERROR_NO_MORE_SEARCH_HANDLES	113	Cannot allocate another search structure and handle.
ERROR_NO_PROC_SLOTS	89	No process slots available.
ERROR_NO_SIGNAL_SENT	205	No signal sent — no process in the command subtree has a signal handler.
ERROR_NO_VOLUME_LABEL	125	No volume label found with DosQFSInfo command.
ERROR_OPEN_FAILED	110	Open/create failed due to explicit fail command.
ERROR_OPLOCKED_FILE	268	Oplocked file.
ERROR_OPLOCK_THREAD_EXISTS	269	Oplock thread exists.
ERROR_OUT_OF_PAPER	28	Printer out of paper.
ERROR_OUT_OF_STRUCTURES	84	Too many redirections.
ERROR_PATH_BUSY	148	Path specified is being used by another process.
ERROR_PATH_NOT_FOUND	3	Path not found.
ERROR_PIPE_BUSY	231	Pipe is busy.
ERROR_PIPE_NOT_CONNECTED	233	Pipe was disconnected by server.
ERROR_PROC_NOT_FOUND	127	Procedure address not found with getprocaddr.
ERROR_PROTECTION_VIOLATION	115	Bad user virtual address.
ERROR_QUE_CURRENT_NAME	329	Current queue name does not exist.
ERROR_QUE_DUPLICATE	332	Duplicate queue name.
ERROR_QUE_ELEMENT_NOT_EXIST	333	Queue element does not exist.
ERROR_QUE_EMPTY	342	Queue is empty.
ERROR_QUE_INVALID_HANDLE	337	Invalid queue handle.
ERROR_QUE_INVALID_NAME	335	Invalid queue name.
ERROR_QUE_INVALID_PRIORITY	336	Invalid queue priority parameter.
ERROR_QUE_INVALID_WAIT	433	NoWait parameter out of bounds.

ERROR_QUE_LINK_NOT_FOUND	338	Queue link not found.
ERROR_QUE_MEMORY_ERROR	339	Queue memory error.
ERROR_QUE_NAME_NOT_EXIST	343	Queue name does not exist.
ERROR_QUE_NOT_INITIALIZED	344	Queues not initialized.
ERROR_QUE_NO_MEMORY	334	Inadequate queue memory.
ERROR_QUE_PREV_AT_END	340	Previous queue element was at end of queue.
ERROR_QUE_PROC_NOT_OWNED	330	Current process does not own queue.
ERROR_QUE_PROC_NO_ACCESS	341	Process does not have access to queues.
ERROR_QUE_PROC_OWNED	331	Current process owns queue.
ERROR_QUE_UNABLE_TO_ACCESS	345	Unable to access queues.
ERROR_QUE_UNABLE_TO_ADD	346	Unable to add new queue.
ERROR_QUE_UNABLE_TO_INIT	347	Unable to initialize queues.
ERROR_READ_FAULT	30	Read fault.
ERROR_RELOC_CHAIN_XEEDS_SEGLIM	201	Relocation chain exceeds segment limit.
ERROR_RING2SEG_MUST_BE_MOVABLE	200	Privilege level 2 segment must be movable.
ERROR_RING2_STACK_IN_USE	207	Privilege level 2 stack in use.
ERROR_SAME_DRIVE	143	Cannot join or substitute a drive to a directory on the same drive.
ERROR_SCS_CALL	364	Call issued by other than sm
ERROR_SCS_INVALID_FUNCTION	424	Caller invalid function.
ERROR_SCS_NOT_SESSION_MGR	425	Caller not session manager.
ERROR_SCS_NOT_SHELL	420	Caller is not shell.
ERROR_SCS_SG_NOTFOUND	419	New session number.
ERROR_SCS_VALUE	365	Value is not for save or restore.
ERROR_SEARCH_STRUC_REUSED	260	DOS mode findfirst/next search structure reused.
ERROR_SECTOR_NOT_FOUND	27	Sector not found.
ERROR_SEEK	25	Seek error.
ERROR_SEEK_ON_DEVICE	132	Application trying to seek on device or pipe.
ERROR_SEM_IS_SET	102	DosCloseSem found semaphore set.
ERROR_SEM_NOT_FOUND	187	Semaphore does not exist.
ERROR_SEM_OWNER_DIED	105	Previous semaphore owner terminated without freeing semaphore.
ERROR_SEM_TIMEOUT	121	Time out occurred from semaphore API function.
ERROR_SEM_USER_LIMIT	106	Semaphore limit exceeded.
ERROR_SGS_NOT_SESSION_MGR	368	Caller not session manager.
ERROR_SHARING_BUFFER_EXCEEDED	36	Sharing buffer overflow.
ERROR_SHARING_VIOLATION	32	Sharing violation.
ERROR_SIGNAL_PENDING	162	Signal already pending.
ERROR_SIGNAL_REFUSED	156	Signal refused.
ERROR_SMG_BAD_ACTION	417	Invalid action specified.
ERROR_SMG_BAD_RESERVE	459	Reserved parameters not zero.
ERROR_SMG_BAD_STATUSREQ	432	Invalid status requested.
ERROR_SMG_GRP_NOT_FOUND	371	Session not found.
ERROR_SMG_INVALID_BOND_OPTION	455	Invalid session bond option.
ERROR_SMG_INVALID_CALL	418	INIT called more than once or invalid session identity.
ERROR_SMG_INVALID_DATA_LENGTH	461	Invalid data length.
ERROR_SMG_INVALID_DEBUG_PARMS	503	
ERROR_SMG_INVALID_ICON_FILE	506	
ERROR_SMG_INVALID_INHERIT_OPT	493	Bad inherit option.
ERROR_SMG_INVALID_PGM_CONTROL	492	Invalid program control.
ERROR_SMG_INVALID_PROGRAM_TYPE	491	Invalid program type.
ERROR_SMG_INVALID_RELATED_OPT	454	Invalid session start related option.
ERROR_SMG_INVALID_SELECT_OPT	456	Invalid session select option.
ERROR_SMG_INVALID_SESSION_ID	369	Invalid session ID.
ERROR_SMG_INVALID_SGID	369	Invalid session identity.
ERROR_SMG_INVALID_START_MODE	453	Invalid session start mode.
ERROR_SMG_INVALID_STOP_OPTION	458	Invalid session stop option.
ERROR_SMG_INVALID_TRACE_OPTION	478	Invalid start session trace indicator.
ERROR_SMG_NOSG	370	No sessions available.
ERROR_SMG_NOT_BASESHELL	431	Caller is not the base shell.
ERROR_SMG_NOT_BOUND	462	Parent not bound.
ERROR_SMG_NO_HARD_ERRORS	481	No session manager hard errors.
ERROR_SMG_NO_SESSIONS	370	No sessions available.

ERROR_SMG_PROCESS_NOT_PARENT	460	Session parent process already exists.
ERROR_SMG_RETRY_SUB_ALLOC	463	Retry request block allocation.
ERROR_SMG_SESSION_NON_SELECT	450	Session is not selectable.
ERROR_SMG_SESSION_NOT_FOREGRND	451	Parent/child session not foreground.
ERROR_SMG_SESSION_NOT_FOUND	371	Session not found.
ERROR_SMG_SESSION_NOT_PARENT	452	Not parent of requested child.
ERROR_SMG_SET_TITLE	372	Title sent by shell or parent cannot be changed.
ERROR_SMG_START_IN_BACKGROUND	457	Session started in background.
ERROR_STACK_IN_HIGH_MEMORY	218	Stack in high memory.
ERROR_SUBST_TO_JOIN	141	Cannot substitute to a joined drive.
ERROR_SUBST_TO_SUBST	139	Cannot substitute to a substituted drive.
ERROR_SWAPIN_FAILED	169	Swap in failed.
ERROR_SWAPIO_FAILED	168	Swap IO failed.
ERROR_SYSTEM_TRACE	150	System trace error.
ERROR_SYS_INTERNAL	328	Internal system error.
ERROR_THREAD_1_INACTIVE	210	Inactive thread.
ERROR_TOO_MANY_MODULES	214	Too many modules.
ERROR_TOO_MANY_MUXWAITERS	152	System limit of 100 entries reached.
ERROR_TOO_MANY_OPEN_FILES	4	Too many open files (no handles left).
ERROR_TOO_MANY_SEMAPHORES	100	User/system open semaphore limit exceeded.
ERROR_TOO_MANY_SEM_REQUESTS	103	Too many exclusive semaphore requests.
ERROR_TOO_MANY_TCBS	155	Cannot create another TCB.
ERROR_TOO_MUCH_STACK	262	Stack request exceeds system limit.
ERROR_TRANSFER_TOO_LONG	222	Transfer is too long.
ERROR_TS_DATETIME	327	Date or time invalid.
ERROR_TS_HANDLE	326	Invalid timer handle.
ERROR_TS_NOTIMER	324	No timers available.
ERROR_TS_SEMHANDLE	323	Invalid system semaphore.
ERROR_TS_WAKEUP	322	Unable to wake up.
ERROR_UNCERTAIN_MEDIA	163	ERROR_I24 mapping.
ERROR_UNC_DRIVER_NOT_INSTALLED	166	Default redir return code
ERROR_UNEXPECTED_SLOT_RETURNED	477	Internal error.
ERROR_VC_DISCONNECTED	240	Session was dropped due to errors.
ERROR_VIOKBD_REQUEST	116	Error on display write or keyboard read.
ERROR_VIO_APTR	351	Invalid pointer to attribute.
ERROR_VIO_ASSOCIATED_DC	499	
ERROR_VIO_ATTR	357	Invalid cursor attribute value.
ERROR_VIO_BAD_CP	469	Invalid code page specified.
ERROR_VIO_BAD_RESERVE	486	Reserved parameter is not zero.
ERROR_VIO_BOTROW	361	Invalid BotRow value.
ERROR_VIO_COL	359	Invalid column value.
ERROR_VIO_CPTR	353	Invalid pointer to column.
ERROR_VIO_DEREGISTER	404	VioDeRegister not allowed.
ERROR_VIO_DETACHED	465	This call disallowed for detached pid.
ERROR_VIO_EXISTING_POPUP	406	Pop-up window on screen (NoWait).
ERROR_VIO_EXTENDED_SG	494	
ERROR_VIO_FONT	467	No font available to support mode.
ERROR_VIO_FUNCTION_OWNED	422	Save/restore already owned.
ERROR_VIO_ILLEGAL_DURING_LOCK	437	Function not allowed during screen lock.
ERROR_VIO_ILLEGAL_DURING_POPUP	430	Function not allowed during pop-up window.
ERROR_VIO_INTERNAL_RESOURCE	479	VIO internal resource error.
ERROR_VIO_INVALID_ASCII_Z	403	Invalid ASCII_Z length.
ERROR_VIO_INVALID_HANDLE	436	Invalid VIO handle.
ERROR_VIO_INVALID_LENGTH	438	Invalid VIO length.
ERROR_VIO_INVALID_MASK	349	Invalid function replaced.
ERROR_VIO_INVALID_PARMS	421	Invalid parameters passed.
ERROR_VIO_IN_BG	429	Function invalid in background.
ERROR_VIO_LEFTCOL	363	Invalid left column value.
ERROR_VIO_LOCK	434	Error returned from Scroll Lock.
ERROR_VIO_LPTR	354	Invalid pointer to length.
ERROR_VIO_MODE	355	Unsupported screen mode.
ERROR_VIO_NA_CP	471	Current display does not support code page.
ERROR_VIO_NOT_PRES_MGR_SG	495	

ERROR_VIO_NO_CP	470	System displays do not support code page.
ERROR_VIO_NO_MODE_THREAD	427	No mode restore thread in SG.
ERROR_VIO_NO_MORE_HANDLES	497	
ERROR_VIO_NO_POPUP	405	Pop-up window not allocated.
ERROR_VIO_NO_SAVE_RESTORE_THD	428	No save/rest thread in SG.
ERROR_VIO_PTR	350	Invalid pointer to parameter.
ERROR_VIO_REGISTER	426	Vio register not allowed.
ERROR_VIO_RETURN	423	Non-destruct return (undo).
ERROR_VIO_RIGHTCOL	362	Invalid right column value.
ERROR_VIO_ROW	358	Invalid row value.
ERROR_VIO_RPTR	352	Invalid pointer to row.
ERROR_VIO_SEE_ERROR_LOG	498	
ERROR_VIO_SHELL_INIT	480	VIO shell initialization error.
ERROR_VIO_SHIELD_OWNED	496	
ERROR_VIO_SMG_ONLY	402	Valid from session manager only.
ERROR_VIO_TOPROW	360	Invalid TopRow value.
ERROR_VIO_TRANSPARENT_POPUP	483	Error during VIO pop-up window.
ERROR_VIO_UNLOCK	367	Screen not previously locked.
ERROR_VIO_USER_FONT	468	User font active.
ERROR_VIO_WAIT_FLAG	366	Invalid wait flag setting.
ERROR_VIO_WIDTH	356	Invalid cursor width value.
ERROR_VOLUME_CHANGED	270	Volume changed.
ERROR_WAIT_NO_CHILDREN	128	DosCwait finds no children.
ERROR_WRITE_FAULT	29	Write fault.
ERROR_WRITE_PROTECT	19	Attempt to write on write-protected diskette.
ERROR_WRONG_DISK	34	Invalid disk change.
ERROR_ZERO_SELECTORS_REQUESTED	488	At least one selector must be requested.
ERROR_ZOMBIE_PROCESS	217	Zombie process.
ERR_TSTDUP	92	Timer service table duplicate.
ERR_TSTOVFL	91	Timer service table overflow.
NO_ERROR	0	No error occurred.
NO_ERROR_MOUSE_NO_DATA	393	No valid data.
NO_ERROR_MOUSE_PTR_DRAWN	394	Pointer drawn.

# Index

## A

- application type 2-243
- ASCII mode 3-29
- attach a pseudo-character device name to an FSD 2-111
- attach a volume to a remote file system 2-111
- attached remote file system, query information from a 2-257
- attributes, file 2-255

## B

- BINARY mode 3-29
- Buffer
  - display video 5-79
  - flush keystroke 3-7
  - get physical video 5-20
  - get video 5-5
  - register as monitor 2-191
  - reset 2-11

## C

- case map 2-16
- characters, global file name 2-56
- Child
  - wait for termination 2-42
- Code Page
  - get process 2-122
  - get video 5-11
  - set 2-320
  - set process 2-344
  - set video 5-62
- Code Segment
  - create cs alias 2-32
- collate table, get 2-120
- constant names 1-1
- control, file system 2-113
- copy file 2-30
- country information 2-124
- Critical Section
  - enter 2-58
  - exit 2-76
  - of execution 2-58

## D

- date, get 2-128
- date, set 2-322
- detach a pseudo-character device name from an FSD 2-111
- detach a volume from a remote file system 2-111
- Device
  - configuration 2-47
  - installed 2-47
  - I/O control 2-49, 2-51
- Device Monitor
  - close connection 2-187
  - open connection 2-188
  - read input 2-189
  - register buffers 2-191

### Device Monitor (*continued*)

- write output 2-193
- device name, character 2-257
- device name, pseudo 2-257
- Directory
  - change current 2-18
  - query current 2-245
- directory information, enumerate 2-59
- directory information, query 2-271
- directory information, set 2-340
- Disk
  - partitionable support 2-231
  - query current 2-247
  - select drive 2-299
- display mode table 5-72
- DosAllocHuge 2-2
- DosAllocSeg 2-5
- DosAllocShrSeg 2-8
- DosBeep 2-10
- DosBufReset 2-11
- DosCallback 2-13
- DosCallNmPipe 2-14
- DosCaseMap 2-16
- DosChDir 2-18
- DosChgFilePtr 2-20
- DosCLIAccess 2-22
- DosClose 2-23
- DosCloseQueue 2-25
- DosCloseSem 2-26
- DosConnectNmPipe 2-28
- DosCopy 2-30
- DosCreateCSAlias 2-32
- DosCreateQueue 2-34
- DosCreateSem 2-36
- DosCreateThread 2-39
- DosCwait 2-42
- DosDelete 2-45
- DosDevConfig 2-47
- DosDevIOCtl 2-49
- DosDevIOCtl2 2-51
- DosDisconnectNmPipe 2-53
- DosDupHandle 2-54
- DosEditName 2-56
- DosEnterCritSec 2-58
- DosEnumAttribute 2-59
- DosErrClass 2-62
- DosError 2-65
- DosExecPgm 2-67
- DosExit 2-73
- DosExitCritSec 2-76
- DosExitList 2-77
- DosFileIO 2-79
- DosFileLocks 2-82
- DosFindClose 2-85
- DosFindFirst 2-86
- DosFindFirst2 2-91
- DosFindNext 2-99
- DosFlagProcess 2-103
- DosFreeModule 2-106
- DosFreeResource 2-108
- DosFreeSeg 2-109
- DosFSAttach 2-111

DosFSCtl	2-113	DosQNmPipeInfo	2-267
DosFSRamSemClear	2-115	DosQNmPipeSemState	2-269
DosFSRamSemRequest	2-117	DosQPathInfo	2-271
DosGetCollate	2-120	DosQSysInfo	2-277
DosGetCp	2-122	DosQueryQueue	2-278
DosGetCtryInfo	2-124	DosQVerify	2-279
DosGetDateTime	2-128	DosRead	2-281
DosGetDBCSEv	2-131	DosReadAsync	2-283
DosGetEnv	2-133	DosReadQueue	2-285
DosGetHugeShift	2-135	DosReallocHuge	2-287
DosGetInfoSeg	2-136	DosReallocSeg	2-289
DosGetMachineMode	2-142	DosResumeThread	2-291
DosGetMessage	2-143	DosRmdir	2-293
DosGetModHandle	2-146	DosR2StackRealloc	2-280
DosGetModName	2-147	DosScanEnv	2-294
DosGetPID	2-148	DosSearchPath	2-296
DosGetPPID	2-151	DosSelectDisk	2-299
DosGetProcAddr	2-153	DosSelectSession	2-300
DosGetPrtY	2-154	DosSemClear	2-301
DosGetResource	2-157	DosSemRequest	2-303
DosGetResource2	2-159	DosSemSet	2-306
DosGetSeg	2-161	DosSemSetWait	2-309
DosGetShrSeg	2-162	DosSemWait	2-313
DosGiveSeg	2-165	DosSendSignal	2-317
DosHoldSignal	2-166	DosSetCp	2-320
DosInsMessage	2-169	DosSetDateTime	2-322
DosKillProcess	2-171	DosSetFHandState	2-324
DosLoadModule	2-174	DosSetFileInfo	2-326
DosLockSeg	2-176	DosSetFileMode	2-331
DosMakeNmPipe	2-177	DosSetFSInfo	2-333
DosMakePipe	2-180	DosSetMaxFH	2-335
DosMemAvail	2-181	DosSetNmPHandState	2-336
DosMkdir	2-182	DosSetNmPipeSem	2-338
DosMkdir2	2-183	DosSetPathInfo	2-340
DosMonClose	2-187	DosSetProcCp	2-344
DosMonOpen	2-188	DosSetPrtY	2-345
DosMonRead	2-189	DosSetSession	2-349
DosMonReg	2-191	DosSetSigHandler	2-352
DosMonWrite	2-193	DosSetVec	2-356
DosMuxSemWait	2-196	DosSetVerify	2-358
DosNewSize	2-200	DosShutdown	2-360
DosOpen	2-201	DosSizeSeg	2-359
DosOpenQueue	2-214	DosSleep	2-361
DosOpenSem	2-215	DosSMRegisterDD	2-364
DosOpen2	2-207	DosStartSession	2-366
DosPeekNmPipe	2-218	DosStopSession	2-373
DosPeekQueue	2-220	DosSubAlloc	2-375
DosPFSActivate	2-222	DosSubFree	2-376
DosPFSCloseUser	2-224	DosSubSet	2-377
DosPFSInit	2-225	DosSuspendThread	2-379
DosPFSQueryAct	2-227	DosTimerAsync	2-381
DosPFSVerifyFont	2-229	DosTimerStart	2-383
DosPhysicalDisk	2-231	DosTimerStop	2-385
DosPortAccess	2-233	DosTransactNmPipe	2-387
DosPtrace	2-235	DosUnlockSeg	2-389
DosPurgeQueue	2-241	DosWaitNmPipe	2-390
DosPutMessage	2-242	DosWrite	2-391
DosQAppType	2-243	DosWriteAsync	2-394
DosQCurDir	2-245	DosWriteQueue	2-396
DosQCurDisk	2-247	Dynamic Link	
DosQFHandState	2-248	free module	2-106
DosQFileInfo	2-250	get module handle	2-146
DosQFileMode	2-255	get module name	2-147
DosQFSAttach	2-257	get procedure address	2-153
DosQFSInfo	2-260	load module	2-174
DosQHandType	2-263		
DosQNmPHandState	2-265		

## E

- enumerate
  - a directory's information 2-59
  - a file's information 2-59
  - a subdirectory's information 2-59
- Environment String
  - get address 2-133
- equipment check 2-47
- error code classification 2-62
- error descriptions
  - base OS/2 calls A-1
- Errors
  - classify codes 2-62
  - processing 2-65
- executable file 2-243

## F

- File
  - change size 2-200
  - delete 2-45
  - find first matching 2-86, 2-91
  - find next matching 2-99
  - lock manager 2-82
  - move 2-194
  - open 2-201, 2-207
  - query information 2-250
  - query mode 2-255
  - query system information 2-260
  - read 2-281
  - read asynchronous 2-283
  - search path for name 2-296
  - set information 2-326
  - set mode/attribute 2-331
  - set system information 2-333
  - write to, asynchronous 2-394
  - write to, synchronous 2-391
- file access, lock 2-79
- file access, unlock 2-79
- file attributes 2-255
- File Handle
  - close 2-23
  - duplicate 2-54
  - query state 2-248
  - set maximum 2-335
  - set state 2-324
- file information, enumerate 2-59
- file information, query 2-271
- file information, set 2-340
- file IO, read 2-79
- file IO, write 2-79
- file lock manager 2-82
- file name, search 2-296
- file pointer, change 2-20
- file system control 2-113
- file system, DosShutdown 2-360
- file, copy 2-30
- file, move or rename a 2-194
- find handle, close 2-85
- Flags
  - set external event 2-103
- Font
  - activate 2-222
  - initialize 2-225
  - query active 2-227
  - verify 2-229

- foreground session, select 2-300
- function descriptions
  - conventions used 1-1
  - notation 1-1

## G

- Get Version Number 2-163
- global editing characters 2-56
- global file name characters 2-56
- global search characters 2-56

## H

- Handle
  - query type 2-263
- hard error processing 2-65

## I

- implicit pointer 1-1
- I/O Privilege
  - disable/enable interrupts 2-22
  - port access 2-233
  - request CLI/STI 2-22
  - request/release access 2-233

## K

- KbdCharIn 3-2
- KbdClose 3-5
- KbdDeRegister 3-6
- KbdFlushBuffer 3-7
- KbdFreeFocus 3-8
- KbdGetCp 3-9
- KbdGetFocus 3-10
- KbdGetHWId 3-11
- KbdGetStatus 3-13
- KbdOpen 3-16
- KbdPeek 3-17
- KbdRegister 3-20
- KbdSetCp 3-22
- KbdSetCustXt 3-24
- KbdSetFgnd 3-25
- KbdSetStatus 3-26
- KbdStringIn 3-29
- KbdSynch 3-31
- KbdXlate 3-32
- Keyboard
  - close 3-5
  - deregister subsystem 3-6
  - free focus 3-8
  - get focus 3-10
  - get status 3-13
  - open 3-16
  - peek character 3-17
  - peek scan code 3-17
  - read character 3-2
  - read character string 3-29
  - read scan code 3-2
  - register subsystem 3-20
  - scan code 3-2
  - set priority 3-25
  - set status 3-26
  - synchronize access 3-31
  - translate scan code 3-32



## L

lock file access 2-79

## M

matching file, find first 2-86, 2-91

matching file, find next 2-99

### Memory

allocate huge 2-2

allocate segment 2-5

allocate shared segment 2-8

change huge size 2-287

free suballocated 2-376

get largest free block 2-181

get shared segment 2-162

set allocated 2-377

suballocate segment 2-375

### Messages

output text to handle 2-242

system 2-143

variable text 2-143

variable text strings 2-169

mode, display table 5-72

module definition file, NAME statement 2-243

MouClose 4-2

MouDeRegister 4-3

MouDrawPtr 4-4

MouFlushQue 4-5

MouGetDevStatus 4-6

MouGetEventMask 4-7

MouGetNumButtons 4-8

MouGetNumMickeys 4-9

MouGetNumQueEI 4-10

MouGetPtrPos 4-12

MouGetPtrShape 4-14

MouGetScaleFact 4-17

MouInitReal 4-19

MouOpen 4-21

MouReadEventQue 4-23

MouRegister 4-26

MouRemovePtr 4-29

### Mouse

get scale factor 4-17

get synchronous access 4-42

initialize DOS mode 4-19

register subsystem 4-26

set event mask 4-33

set scale factor 4-40

### Mouse Device

close 4-2

open 4-21

set status 4-31

### Mouse Pointer

draw 4-4

remove 4-29

set position 4-35

set shape 4-37

### Mouse Queue

flush 4-5

read event 4-23

MouSetDevStatus 4-31

MouSetEventMask 4-33

MouSetPtrPos 4-35

MouSetPtrShape 4-37

MouSetScaleFact 4-40

MouSynch 4-42

move a file or subdirectory 2-194

## N

name editing 2-56

NAME statement 2-243

### Named Pipe

close 2-53

connect 2-28

create 2-177

disconnect 2-53

instance availability 2-390

make 2-177

operations, query 2-269

peek 2-218

perform transaction 2-387

procedure call 2-14

query 2-265

query info 2-267

read 2-218

set handle state 2-336

set semaphore 2-338

wait 2-390

notation conventions 1-1

NULL 1-1

## O

Open a File 2-201, 2-207

## P

### Pipes

create 2-180

pointer, implicit 1-1

power off, DosShutdown 2-360

privilege level 2 stack 2-280

### Process

delay execution 2-361

get ID 2-148

get parent ID 2-151

get priority 2-154

routine list termination 2-77

set code page 2-344

set priority 2-345

termination 2-171

processor mode 2-142

### Program

debug interface 2-235

execution 2-67

exit 2-73

pseudo device name 2-257

pseudo-character device name 2-111, 2-257

pseudo-character device name attached to an FSD,  
query 2-257

## Q

query a directory's information 2-271

query a file's information 2-271

query a subdirectory's information 2-271

query info about a pseudo-character device attached to  
an FSD 2-257

query info about an attached remote file system 2-257

query system variables 2-277

### Queue

close 2-25

## Queue (continued)

- make queue 2-34
- open 2-214
- peek 2-220
- purge 2-241
- query size 2-278
- query, verify setting 2-279
- read 2-285
- write to 2-396

## R

- read file IO 2-79
- register session switch notification 2-364
- rename a file or subdirectory 2-194
- reset, buffers 2-11
- resource 2-108, 2-159
- resource segment 2-157
- return code classification 2-62
- routine list 2-77

## S

- scan code, translate 3-32
- Segment
  - access shared segment 2-161
  - allocate 2-5
  - allocate shared 2-8
  - change size 2-289
  - free suballocated memory 2-376
  - get shared segment 2-162
  - give access to 2-165
  - lock 2-176
  - scan environment 2-294
  - suballocate memory 2-375
  - system variable address 2-136
  - unlock 2-389
- segment size 2-359
- Semaphore
  - clear 2-115
  - clear (release) 2-301
  - close, system 2-26
  - create system semaphore 2-36
  - open, system 2-215
  - request 2-117, 2-303
  - set owned 2-306
  - set/wait for clear 2-309
  - wait N to clear 2-196
  - wait to clear 2-313
- Session
  - start 2-366
  - stop 2-373
- session status, set 2-349
- set a directory's information 2-340
- set a file's information 2-340
- shift count 2-135
- Shutdown 2-360
- signal focus 2-353
- Signal Handler
  - set 2-352
- Signals
  - disable/enable 2-166
  - send control break 2-317
  - send control C 2-317
- speaker sound 2-10
- Subdirectory
  - create 2-182, 2-183

## Subdirectory (continued)

- remove 2-293
- subdirectory information, enumerate 2-59
- subdirectory information, query 2-271
- subdirectory, move or rename a 2-194
- system variables, query 2-277

## T

- Thread
  - create 2-39
  - restart 2-291
  - suspend execution 2-379
- time delay, start Async 2-381
- Timer
  - start 2-383
  - stop 2-385
- time, get 2-128
- time, set 2-322
- TRC\_\* values 2-239

## U

- unlock file access 2-79

## V

- Vector
  - DBCS 2-131
  - environmental 2-131
- vector, set 2-356
- verify switch, set 2-358
- version number, get 2-163
- Video
  - ANSI status 5-4
  - configuration 5-7
  - deallocate pop-up 5-3
  - deregister subsystem 5-2
  - get font 5-15
  - get state 5-22
  - register subsystem 5-43
  - set ANSI 5-61
  - set code page 5-62
  - set font 5-67
  - set state 5-74
- Video Cursor
  - get position 5-12
  - get type 5-13
  - set position 5-64
  - set type 5-65
- Video Mode
  - display 5-17
  - restore 5-30
  - restore wait 5-32
  - set display 5-69
- Video pop-up
  - allocate 5-34
- Video Read
  - character string 5-41
  - char/attr string 5-39
- Video Screen
  - lock 5-50
  - print 5-37
  - print toggle 5-38
  - save redraw undo 5-46
  - save redraw wait 5-48
  - scroll down 5-52

#### Video Screen (*continued*)

- scroll left 5-54
- scroll right 5-56
- scroll up 5-58
- unlock 5-60

#### Video Write

- char string w/attrib 5-84
- character string 5-82
- char/attr string 5-80
- N attributes 5-86, 5-88
- N characters 5-88, 5-90
- TTY string 5-92

VioDeRegister 5-2

VioEndPopUp 5-3

VioGetAnsi 5-4

VioGetBuf 5-5

VioGetConfig 5-7

VioGetCp 5-11

VioGetCurPos 5-12

VioGetCurType 5-13

VioGetFont 5-15

VioGetMode 5-17

VioGetPhysBuf 5-20

VioGetState 5-22

VioGlobalReg 5-27

VioModeUndo 5-30

VioModeWait 5-32

VioPopUp 5-34

VioPrtSc 5-37

VioPrtScToggle 5-38

VioReadCellStr 5-39

VioReadCharStr 5-41

VioRegister 5-43

VioSavRedrawUndo 5-46

VioSavRedrawWait 5-48

VioScrLock 5-50

VioScrollDn 5-52

VioScrollLf 5-54

VioScrollRt 5-56

VioScrollUp 5-58

VioScrUnLock 5-60

VioSetAnsi 5-61

VioSetCp 5-62

VioSetCurPos 5-64

VioSetCurType 5-65

VioSetFont 5-67

VioSetMode 5-69

VioSetState 5-74

VioShowBuf 5-79

VioWrtCellStr 5-80

VioWrtCharStr 5-82

VioWrtCharStrAtt 5-84

VioWrtNAttr 5-86

VioWrtNCell 5-88

VioWrtNChar 5-90

VioWrtTTY 5-92

## W

write file IO 2-79

IBM United Kingdom  
International Products Limited  
PO Box 41, North Harbour  
Portsmouth, PO6 3AU  
England

Printed in Denmark  
by Lassen Offset

**IBM**